4. Interface Scenarios and Scenario Primitives

4.1 Interface Scenarios

4.1.1 Introduction

This section outlines a series of eleven system level scenarios. The first three scenarios were generated during Release A and have been modified to reflect upgrades in Release B. The next eight scenarios have been added in support of Release B exclusively in which the last three scenarios have been added to demonstrate the FOS/SDPS interoperability. The purpose of these scenarios is to validate the class category interfaces (a CSCI consists of one or more class categories). These scenarios illustrate the major interactions between class categories and verify that CSCIs are correct. The scenarios contained herein are different than those found in the DID 305 subsystem documents, as they focus on the interfaces between class categories and not the internal workings of class categories.

4.1.2 Approach

For each scenario, the scenario's purpose is stated, the preconditions required for a particular scenario, and define the start and end points. A detailed step-by-step description of the scenario follows each overview. The detailed scenario description is further supported by a table outlining each step in the scenario. In general, a major step in a scenario corresponds to a scenario primitive. In such cases, the scenario description table contains a reference to the scenario primitive. Finally, the scenario primitives are detailed in the form of an event trace diagram.

These interface scenarios represent the most common threads of operation performed by ECS.

4.1.3 Scenarios Overview

The eleven scenarios exercise a broad range of CSCI interfaces included in the ECS Release B system and validate the key architectural interfaces.

The scenarios covered in this section are:

1. *General Data Ingest* – Demonstrating the steps required for the ingestion of data, L0 and higher, from an external facility and the Ingest CSCI interfaces.
2. *User Access* – Demonstrating the search, browse, and acquire product mechanisms.
3. *Standard Production* – Demonstrating the nominal Release A data production strategy with enhancements for Release B.
4. *Cross-DAAC Search* - Demonstrating search capability across multiple DAAC sites.
5. *On-Demand Processing* - Demonstrating the request for data processing on-demand.
7. *Data Acquisition Request (DAR) Processing* - Demonstrating the capability to request instrument data acquisition, and subsequent generation of higher level products.
8. *Billing and Accounting Scenario* - Demonstrating the capability to update accounting information when a product is shipped and when a product's price changes.

9. *FOS/SDPS Interaction Scenario* - three scenarios demonstrating the interoperability of FOS and SDPS to submit, query, and acquire data.
10. *FOS Data Advertisement Scenario* - use of advertisement service to advertise data availability schedules
11. *FOS-INGEST Interaction Scenario* - use of the Ingest CSCI to put FOS data into Science Data Server.

4.1.4 Scenario Description Tables

The tables in the following sections are divided into six columns containing the following information:

Column 1: Step Number

This is the sequence number of the step in the scenario.

Column 2: EName

A short name to identify the event described by the scenario step.

Column 3. Primitive Reference

A reference to the scenario primitive(s) in Section 4.2. For each scenario primitive, an event trace diagram is provided which identifies the objects and exact sequence of operations used on those objects to implement the scenario step. Each scenario primitive has a reference name consisting of the prefix ***SP*** followed by a unique number. For example, the scenario primitive which provides the detailed sequence of events necessary to acquire data granules from the data server is referred to as SP 11. The unique number is embedded in the event trace diagram name.

Column 4: Interface Client

Identification of the CSCI acting as the client in the interface (i.e., requesting a service from the server).

Column 5: Interface Server

Identification of the CSCI acting as the server in the interface (i.e., providing a service to the client).

Column 6: Description

A brief description of the scenario step.

4.1.5 Scenario 1 - General Data Ingest

4.1.5.1 Purpose

The purpose of this scenario is to demonstrate how data is ingested into the ECS. This scenario also illustrates the following CSCI interfaces:

1. The use of the subscription notification mechanism to enable Planning CSCI (PLANG) to receive notification when the L0 data is available for processing in ECS.
2. The interface provided by the Science Data Server CSCI (SDSRV) from the Ingest CSCI to insert L0 data.

3. The use of a gateway to enable the Ingest CSCI (INGST) to communicate with and access data from external systems in a standard way.

4.1.5.2 Start and End Conditions

The scenario starts upon the submittal of a subscription via the PLANG CSCI user interface to allow PLANG to receive notification when the L0 or higher data has been successfully inserted into ECS.

The scenario is terminated by the Data Delivery Notification (DDN) message from INGST to the external facility that produced the L0 or higher data.

4.1.5.3 Preconditions

1. An operator has submited a subscription via the PLANG CSCI user interface. This subscription is submitted against the SDSRV CSCI that will interact with the INGST CSCI to archive the L0 or higher data.
2. INGST had queried the Ingest data base to obtain a UR for the appropriate SDSRV, STMGT, and INGST preprocessing server for the Ingest data type being processed. The Ingest data base has been preloaded with appropriates UR's (**see SP 48_B for details**).

4.1.5.4 Scenario Description

1. INGST receives a Data Availability Notice (DAN) from Ingest gateway (IGTWY) on behalf of an external system, then responds with Data Availability Acknowledgment (DAA), and pulls the data. For L0 data ingest from EDOS, the following steps are performed instead:
   a) EDOS pushes file to INGEST staging area
   b) EDOS sends PDS Delivery Record (DAN) to ECS
   c) ECS ingests data using next steps in scenario in the same fashion
2. INGST loads the L0 or higher data and metadata to a local staging disk.
3. INGST pre-processes the L0 or higher data against the metadata to ensure data values are valid and the data is formatted correctly.
4. INGST requests the SDSRV CSCI to insert the L0 and corresponding metadata or higher data. SDSRV creates an archive and notifies INGST that the archive has been successfully created. NOTE: INGST reuses the SDSRV code on the INGST hardware. From a software viewpoint, the interface is with the SDSRV.
5. SDSRV notifies INGST that the L0 and metadata has been successfully created.
6. SDSRV notifies PLANG that the L0 or higher data is available.
7. INGST sends a Data Delivery Notice (DDN) to notify the external facility of a successful transfer. The external system responds with a Data Delivery Acknowledgment (DDA). For EDOS, the ECS sends a PDS Delivery Verification instead.

This scenario finishes here. ECS would subsequently go into Standard Production as outlined in Scenario 3.

Table 4-1 gives a detailed account of this scenario.

Table 4-1. Scenario 1 General Data Ingest

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Receive DAN	SP 45	IGTWY	INGST	Ingest receives Data Availability Notice (DAN) from Ingest gateway on behalf of an external system, then pulls file using protocol shown in SP45 (For EDOS see 4.1.5.4).
2	Data Stage	SP 4	INGST	STMGT	Load L0 or higher data to local staging disk (normal staging operation).
3	Data Check	SP 2	INGST	SDSRV	Validate L0 or higher against metadata to ensure values are in range, etc.
4	Insert Data	SP 12	INGST	SDSRV	INGST requests SDSRV to insert data. SDSRV creates archive and notifies INGST that archive has been successfully created.
5	Insert Metadata	SP 12	INGST	SDSRV	INGST requests SDSRV to insert metadata. SDSRV creates archive and notifies INGST that archive has been successfully created.
6	Notify subscribers	SP 8	SDSRV	PLANG	Notify PLANG that L0 or higher data is available.
7	Send DDN	SP 45	IGTWY	INGST	Send Data Delivery Notice to notify external facility of successful transfer.

4.1.6 Scenario 2 - User Access

4.1.6.1 Purpose

The purpose of this scenario is to present the steps that a user would typically perform to search, browse, and acquire science data. This scenario also shows how a user can issue multiple queries asynchronously (i.e., additional queries can be made even while a previous query is in progress). In addition, this scenario illustrates how a user can save the combined results from two separate queries for use in a subsequent session. The following CSCI interactions are included:

1. The interaction between the Client Subsystem (CLS) and *an identified* Science Data Server CSCI (SDSRV).
2. The basic retrieval operations provided by SDSRV: Search, Browse, and Acquire.

4.1.6.2 Start and End conditions

The scenario starts with the user starting ESST from his desktop workstation and terminates when the user the user invokes ftp to acquire selected data granules.

4.1.6.3 Preconditions

1. Advertising has been populated with the appropriate advertisements by data server.
2. The interface for the WWW, including searching and browsing advertisements, and the retrieval of HTML documents is OTS.
3. The user is already logged on to ECS via the Release B Client Subsystem CSCI (CLS).
4. The user, via a WWW browser, has accessed the Advertising Service CSCI (ADSRV) and provided a search criteria containing keywords which describe a given topic (**see SP 48_B for details**).
5. The ADSRV has returned a list of advertisements of available data and services which match the selection criteria.
6. The user has activated a hyper link referencing guide data on a particular data product.
7. The Document Data Server CSCI (DDSRV) has retrieved and sent the guide document to the user's WWW browser (**see SP 20 for details**).
8. The Distributed Information Manager CSCI (DIMGR) can directly access the identified SDSRV without going through the Local Information Manager CSCI (LIMGR).

4.1.6.4 Scenario Description

1. The user, from his desktop, starts the Earth Science Search Tool (ESST) which initializes its available attribute set and valids list by querying the Data Dictionary CSCI (DDICT) after verifying latest date/time version against locally stored file.
2. ESST initialization retrieves from Advertising Server (ADSRV) CSCI service availability data.
3. The user creates a search using the ESST GUI. When the user submits a search from the ESST, which is sent to the SDSRV, the ESST determines the routing of the query from the DDICT (either by using information already retrieved or by querying the DDICT again).
4. The user uses the ESST to submit a search, with spatial and temporal criteria, to the SDSRV.
5. From the ESST, the user submits a second search which goes to the SDSRV. The ESST computes the routing information based on the search entered by the user and collection-data server mapping it obtained from the DDICT.
6. The user via the ESST issues another search to the SDSRV.
7. The SDSRV notifies the user that the first search is complete.
8. The CLS ESST loads the result set from the SDSRV into the ESST result window.
9. The ESST issues the browse request to the SDSRV which supplied the query results.
10. The SDSRV notifies the user that the second search is complete.
11. The CLS ESST loads the second result set from the SDSRV into the ESST result window.

12. The user reviews the results of the second search, requests graphical and coverage displays, and requests some browse images. The ESST issues the browse request to the SDSRV which supplied the query results.
13. The user issues an ACQUIRE request to retrieve some data granules corresponding to one or more of the retrieved browse images from the SDSRV archives. SDSRV locates the data granules and sends an e-mail message to the user. The message contains the information necessary for the SDSRV to send the data to the user.

Table 4-2 gives a detailed account of this scenario.

Table 4-2. Scenario 2 User Access (1 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	ESST initialization	SP 51_B	CLS	DDICT	ESST is populated with valid values from the DDICT, i.e. attribute names, type, and collection to server mapping
2	ESST initialization	SP 52_B	CLS	ADSRV	ESST retrieves from ADSRV advertisement URs and whether they are publicly or privately accessible.
3	ESST search submission	SP 9	CLS	SDSRV	A user search is submitted to the SDSRV.
4	Search	SP 9	CLS	SDSRV	User queries SDSRV for inventory of data that matches given search criteria (e.g., spatial search, sub select, etc.). SDSRV Query is executed and results are returned as ESDT object references.
5	ESST search submission	SP 9	CLS	SDSRV	A user search is submitted to the SDSRV.
6	Search	SP 9	CLS	SDSRV	User queries SDSRV for inventory of data that matches given search criteria (e.g., spatial search, sub select, etc.). SDSRV Query is executed and results are returned as ESDT object references.
7	Query 1 Search complete	(last three events of SP 9)	CLS	SDSRV	User is notified that the search is complete.
8	Inspect Result Set	SP 55_1B	CLS	SDSRV	The CLS retrieves the metadata from the SDSRV.
9	Browse 1	SP 29	CLS	SDSRV	User reviews results of query 1 on a map overly and requests some browse images. Browse request is executed by SDSRV and results are returned.

Table 4-2. Scenario 2 User Access (2 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
10	Query 2 Search complete	(last three events of SP 9)	CLS	SDSRV	User is notified that the search is complete.
11	Inspect Result Set 2	SP 55_1B	CLS	SDSRV	The CLS inspects the validity of the second result set.
12	Browse 2	SP 29	CLS	SDSRV	User reviews results of query 2 on a map overly and requests some browse images. Browse request is executed by SDSRV and results are returned.
13	Acquire Data	SP 11	CLS	SDSRV	CLS request acquisition of data. The acquisition request is executed by SDSRV. User is notified by e-mail where the data is located.

4.1.7 Scenario 3 - Standard Production

4.1.7.1 Purpose

This scenario describes the interaction between the Processing CSCI (PRONG), the Production Planing CSCI (PLANG), and the data server subsystem during a standard production run. The processing logic is also applicable to all other types of production, but the planning logic will differ for on-demand processing and reprocessing. Reprocessing is functionally very similar to stand production, but is planned differently. These differences are internal to PLANG and do not enhance our understanding of the interfaces.

PRONG provides schedule management and process control necessary to manage the execution of an entire job stream. PLANG manages data availability dependencies. Therefore, the scenario includes subscription notification of data availability, but does not include the detailed process on the notification of program execution and termination since it is done via database updates. As soon as the data dependencies are met, PLANG provides job information to COTS for execution by PRONG.

PRONG provides the capability to stage input data well in advance of the start of the PGE that requires it. This technique, called predictive staging, attempt to stage data at the optimal time such that production throughput is maximized, and the staging disk is used efficiently. In addition, input data may be retrieved from the Science Data Server CSCI (SDSRV) at a remote DAAC. This is an inherent feature of the SDSRV interface, since it uses the Distributed Object Framework (DOF) provided by the Communications Services Subsystem (CSS).

4.1.7.2 Start and End Conditions

The scenario starts with an operator activating subscriptions to notify PLANG when input data (L0 and ancillary non-L0 data) required for processing becomes available. The scenario terminates with the successful archival of the production outputs.

4.1.7.3 Preconditions

ECS is required to notify science teams whenever a new plan is put into production. In order to satisfy this requirement, the science teams must subscribe to updates to the appropriate DAAC's production plan.

4.1.7.4 Scenario Description

1. PLANG automatically submits subscriptions to be notified when required input data is available for processing.
2. Operations uses a user interface provided by PLANG to active a candidate production plan.
3. PLANG publishes the active plan by archiving a description of the plan as an HTML file in the Document Data Server CSCI (DDSRV). All parties interested in viewing plans must previously subscribe to the appropriate DAAC DDSRV.
4. Sometime later, SDSRV notifies PLANG that L0 data required to execute the plan is available for processing.
5. PLANG verifies the UR for the required L0 data from SDSRV.
6. Sometime later, SDSRV notifies PLANG that ancillary data required to execute the plan is available for processing. This fulfills all outstanding Data Dependencies.
7. PLANG verifies the UR for the required L0 data from SDSRV.
8. PLANG releases the Data Processing Request (DPR) job to active job schedule.
9. If the PGE is not available, PRONG acquires the PGE from SDSRV using standard SDSRV acquire operation.
10. PRONG requests the SDSRV to retrieve the L0 data from the archives using the standard SDSRV Acquire operation. SDSRV copies the data to staging disks allocated to Processing. SDSRV notifies PRONG when the required data is retrieved from the archives. PRONG may predictively stage the L0 and ancillary data prior to actual execution of the DPR.
11. PRONG acquires the ancillary data from the SDSRV in the same manner as L0 data was acquired
12. PRONG starts the execution of a Product Generation Executable (PGE) job. This interface is internal to the COTS, but is shown here to clarify that the PGE must run to produce data outputs.
13. PRONG issues a standard SDSRV Insert request to store the PGE outputs in an SDSRV archive.

Table 4-3 gives a detailed account of this scenario.

Table 4-3. Standard Production (1 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Submit Subscriptions	SP 7	PLANG	SDSRV	PLANG activates subscriptions to enable notification to PLANG of data arrival
2	Activate Candidate Plan	SP 25	PLANG	PRONG	Planning activates a candidate processing plan.
3	Publish DAS	SP 46 (a DAS is inserted into the archive in the same manner as any document)	PLANG	DDSRV	PLANG creates a metadata file describing the plan. This file is reference by a hyper link in a HTML document.
4	Receive Subscription Notification	SP 8	SDSRV	PLANG	SDSRV notifies PLANG that required L0 data is available.
5	Verify UR	SP 55_2B	PLANG	SDSRV	Inspect the subscription notification containing a UR.
6	Receive Subscription Notification	SP 8	PLANG	SDSRV	SDSRV notifies PLANG that the required ancillary data is available.
7	Verify UR	SP 55_1B	PLANG	SDSRV	Inspect the subscription notification containing a UR
8	Release DPR Job	SP 26	PLANG	PRONG	PLANG release DPR to active job schedule.
9	Acquire PGE	SP 11	PRONG	SDSRV	Acquire the PGE needed for production from the SDSRV.
10	Acquire L0 Data for Processing	SP 11	PRONG	SDSRV	Processing requests from SDSRV to retrieve L0 data from the archives. This data can be staged in a predictive fashion. SDSRV retrieves L0 data from archive. PRONG receives confirmation.

Table 4-3. Standard Production (2 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
11	Acquire Other Data for Processing	SP 11	PRONG	SDSRV	Processing requests SDSRV to retrieve non-L0 data from archives. SDSRV retrieves non-L0 data from archive. PRONG receives confirmation.
12	Run PGEJob	COTS	PRONG COTS	PRONG COTS	PGE is a component to the job stream managed by the PRONG COTS.
13	Archive Outputs	SP 12	PRONG	SDSRV	PGE output data is stored in SDSRV archive. PRONG receives confirmation.

4.1.9 Scenario 4 - Cross-DAAC Search

4.1.9.1 Purpose

The purpose of this scenario is to demonstrate how ECS performs a search across multiple DAACs. This scenario is based on the following example:

A user creates a query at the client asking for all SST and humidity data for the Atlantic Ocean between Oct. 1, 1990 and Oct. 31, 1994. Assume that this query goes to a Distributed Information Manager CSCI (DIMGR); the DIMGR will use two different data servers to get the data (SDSRV A and B). A more complicated case would have the DIMGR access one or more Distributed Information Manager CSCIs (DIMGR) which would then access one or more data servers, the DIMGR knows what Information Managers (LIMGR or Data Server) it can access. The DIMGR would choose the correct Information Manager based on optimizations for the given query. The LIMGR would primarily do the same steps as the DIMGR as outlined in this scenario.

4.1.9.2 Start and End Conditions

The scenario starts upon the client's submittal of a query using the Earth Science Search Tool (ESST). The scenario is terminated when the DIMGR combines sub query results from SDSRVs A and B, and then returns the results to the Release B Client.

4.1.9.3 Preconditions

1. The user uses the ESST to provide a search criteria.
2. The Science Data Server (SDSRV) is accessible directly to the DIMGR, so the query does not go through the LIMGR.

4.1.9.4 Scenario Description

1. The ESST initializes its available attribute set and valids list by querying the Data Dictionary CSCI (DDICT) after verifying latest date/time version against locally stored file.
2. ESST establishes a connection to the DIMGR. ESST then submits the search request to the DIMGR using the session created.

3. DIMGR parses the query and determines that subqueries should be sent to two different SDSRVs. The DIMGR uses the ADSRV service to get a UR for each SDSRV instance.
4. DIMGR creates a session to SDSRV A using the UR provided by the ADSRV and submits a sub-query to SDSRV A.
5. DIMGR creates a session to SDSRV B using the UR provided by the ADSRV and submits a sub-query to SDSRV B.
6. SDSRV A sends DIMGR the results for inspection.
7. SDSRV B sends DIMGR the results for inspection.
8. After DIMGR merges the results, the client is notified, iterates through the results, and retrieves the data.

Table 4-5 gives a detailed account of this scenario.

Table 4-5. Cross DAAC Search (1 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	ESST start up	SP 51_B	CLS	DDICT	ESST check the collection time and date stamps and checks against the DDICT.
2	Submit query to DIMGR	SP 52_B	CLS	DIMGR	ESST establishes a connection to the DIMGR. The ESST then submits the search request to the DIMGR using the session created. DIMGR parses the query.
3a	Get UR for SDSRV A	SP 48_B	DIMGR	ADSRV	The DIMGR uses the ADSRV service to get the UR for SDSRV A.
3b	Get UR for SDSRV B	SP 48_B	DIMGR	ADSRV	The DIMGR uses the ADSRV service to get the UR for SDSRV B.
4	Create SDSRV session and submit sub queries at SDSRV A	SP 9	DIMGR	SDSRV	DIMGR creates a session to SDSRV A using the UR provided by the ADSRV and submits sub queries to SDSRV A.
5	Create SDSRV session and submit sub queries at SDSRV B	SP 9	DIMGR	SDSRV	DIMGR creates a session to SDSRV B using the UR provided by the ADSRV and submits sub queries to SDSRV B.

Table 4-5. Cross DAAC Search (2 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
6	DIMGR inspects SDSRV A's result set.	SP 55_2B	DIMGR	SDSRV	SDSRV A sends the results to DIMGR for inspection.
7	DIMGR inspects SDSRV B's result set	SP 55_2B	DIMGR	SDSRV	SDSRV B sends the results to DIMGR for inspection
8	Client is notified and retrieves the data	SP 54_B	CLS	DIMGR	After DIMGR merges the results, the client is notified, iterates through the results, and retrieves the data.

4.1.10 Scenario 6 - On-Demand Processing

4.1.10.1 Purpose

The Science Data Server (SDSRV) CSCI provides the capability to create ESDTs that offer services to create higher level data products on-demand. This scenario demonstrates the On-Demand Processing Request (OPR) interface between the SDSRV CSCI and the Planning (PLANG) CSCI to provide this capability.

4.1.10.2 Start and End Conditions

The scenario starts with a user acquiring an ECS product and ends with the distribution of a data product created on demand.

4.1.10.3 Preconditions

1. The user has previously executed a search for this product (see Scenario 2, User Access) and has saved the search results in order to facilitate subsequent orders of this product. The granule is represented in the metadata using the "Virtual Metadata" mechanism.
2. The user is authorized to submit on-demand processing requests (OPR).
3. The user has not exceeded authorized processing limits.
4. The dependent data products for the OPR exist in the SDSRV archive.
5. The user issues an Acquire request to retrieve some data granules (on demand).
6. Using the CLS, the user supplies runtime parameters for processing the requested granules.

4.1.10.4 Scenario Description

1. SDSRV submits an OPR to the Planning CSCI (PLANG) and PLANG validates the OPR. PLANG validates the OPR by executing a search to verify required input data exists.

2. PLANG submits a DPR to the Processing CSCI (PRONG) to acquire the dependent data sets and run the PGE to create the requested data (see Scenario 3).
3. PLANG returns status, including processing time and resource usage estimates to SDSRV.
4. PLANG returns a UR for the requested data to SDSRV.
5. The SDSRV CSCI requests the Data Distribution (DDIST) CSCI to distribute the data to the user. The data is staged to the pull disk, the user is notified via e-mail that the data is available. The user executes an ftp get to acquire the on-demand product (see Scenario 2).

Table 4-6 gives a detailed account of this scenario.

Table 4-6. On Demand Processing

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Submit OPR & PLANG Validation	SP 49_B	SDSRV	PLANG	SDSRV submits an on-demand processing request to PLANG. PLANG validates request. As part of validation, PLANG searches for the input data.
2	Submit DPR	Last event in SP 25	PLANG	PRONG	PLANG submits DPR to PRONG to acquire dependent data sets and generate requested data product.
3	Notify SDSRV	SP 50_B	SDSRV	PLANG	PLANG notifies SDSRV of status, including processing time and resource usage.
4	Notify SDSRV	SP 50_B	SDSRV	PLANG	PLANG notifies SDSRV that data is available.
5	Distribute data to user	SP 11	SDSRV	DDIST	SDSRV requests distribution of data. The user is notified via e-mail and executes an ftp get to acquire the data.

4.1.11 Scenario 7 - Data Acquisition Request Processing

4.1.11.1 Purpose

The system interface between ECS and the Advanced Spaceborn Thermal Emission and Reflection Radiometer (ASTER) Ground Data System (GDS) supports the capability for ECS science users to submit Data Acquisition Requests (DAR) for the ASTER instrument. Once a DAR is accepted and successfully executed, ASTER GDS will send L1a and L1b data to ECS. When an ECS user submits a DAR, he/she will typically request that one or more higher level ASTER products (Level 2) be generated by ECS upon the arrival and insertion of the Level 1b data.

The purpose of this scenario is to demonstrate how a DAR is submitted and processed within ECS, and how the higher level products are generated. This scenario reuses portions of the following scenarios:

1. Scenario 2 - User Access
2. Scenario 1 - Ingest of Higher Level Products
3. Scenario 6 - On-Demand Processing

4.1.11.2 Preconditions

1. The user is authorized to submit DARs.
2. ASTER GDS provides DAR Client application.
3. ASTER GDS returns a DAR ID to ECS.
4. Several ASTER L1a and L1b scenes will be ingested into ECS as a result of a single DAR.
5. An ECS user may request multiple ASTER products to be generated by ECS each time L1b ASTER data arrives.
6. The DAR ID and identifying DAR parameters are included in the L1 metadata to allow ECS to match a DAR to a specific set of data. *(Refer to document# 209-CD-002-003, ICD Between ECS and ASTER.)*

ECS will distribute individual higher order products to the user as they become available, rather than distribute all requested products at one time

4.1.11.3 Start and End Conditions

The scenario starts with the creation of a DAR, and submittal to ASTER GDS using the Client Subsystem (CLS), and terminates when the user invokes ftp to acquire selected data granules.

4.1.11.4 Scenario Description

1. The user creates a DAR for ASTER L1b data using the ASTER provided DAR tool integrated with CLS.
2. The user specifies several L2 ASTER products to be generated upon the arrival of the L1b data.
3. Using the CLS, the user supplies runtime parameters for processing each L2 product requested.
4. The user submits the DAR and a unique DAR ID is returned.
5. The CLS creates a subscription against the insertion of L1b data to the Science Data Server CSCI (SDSRV) for each L2 product requested. Each subscription contains the DAR ID and the following instructions:

   Event: L1b inserted   (Event that triggers the actions)

   Actions: Search L1b metadata for DAR ID = XX (and/or other metadata values that uniquely identify the data)

   Process data to L2 using runtime inputs provided

   Notify user and distribute the L2 data

6. The Ingest CSCI (INGST) ingests the ASTER L1b and inserts it into the SDSRV archive (see Scenario 4). This event triggers the actions specified in the subscriptions created with the DAR
7. SDSRV determines that the L1b data corresponds to the correct DAR ID.
8. To process the data to L2, SDSRV submits an on-demand processing request (ODPR) to the Planning CSCI (PLANG). PLANG submits a Data Processing Request (DPR) to the Data Processing CSCI (PRONG). PRONG executes the DPR to acquire the L1b data, and run a PGE to create the L2 data which is returned (via a Universal Reference (UR)) to SDSRV (see Scenario 6 - On-demand Processing Request).
9. SDSRV requests the Data Distribution CSCI (DDIST) to distribute the L2 data to the user. The L2 Data is staged to the pull disk, the user is notified via e-mail that the data is available. The user executes an ftp get to acquire the L2 product.

Table 4-7 gives a detailed account of this scenario.

Table 4-7. Data Acquisition Request Processing (1 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Input DAR parameters	GUI	User	CLS	User creates DAR using DAR client tool.
2	Select higher level (L2) ASTER products	SP 48_B	CLS	ADSRV	DAR Client queries ADSRV to find possible ASTER L2 products. User selects L2 products to be generated upon arrival of ASTER L1b data.
3	Input Runtime Parameters	GUI	User	CLS	User inputs run time parameters for each selected L2 product.
4	Submit DAR & get DAR ID	Refer to document# 209-CD-002-003, ICD Between ECS and ASTER GDS.	CLS	ASTER GDS	The user submits the DAR to ASTER GDS. The ASTER GDS returns the unique DAR ID to the CLS.
5	Submit Subscription(s)	SP 7	CLS	SDSRV	The CLS creates a subscription to SDSRV against the insertion of ASTER L1b for each selected L2 product.

Table 4-7. Data Acquisition Request Processing (2 of 2)

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
6	Ingest and insert ASTER L1b	Scenario 1 - Ingest of Higher Level Products	INGST	SDSRV	The ASTER L1a, L1b data and metadata is ingested and inserted into the SDSRV archive. This event triggers the actions associated with the L1b subscriptions.
7	Create and Execute On-Demand Processing Request	Scenario 6 - On-Demand Processing	SDSRV	PDPS	SDSRV creates an ODPR to PLANG which, in turn submits a DPR to PRONG to create the L2 product. Upon successful completion of the DPR, SDSRV is notified and given a UR for the L2 data.
8	Distribute L2 data to user.	SP 11	SDSRV	DDIST	SDSRV requests distribution of L2 data. The user is notified via e-mail and executes an ftp get to acquire the data.

4.1.12 Scenario 8 - Billing And Accounting

4.1.12.1 Purpose

The purpose of this scenario is to demonstrate how Billing and Accounting will handle requests for pricing a data product request and checking the user's available balance to determine whether the product can be ordered by the science user.

Part of the User Access Scenario, this scenario is actually a subscenario that illustrates price checking, balance checking, and user account updates.

This scenario also illustrate the following CSCI interfaces:

1. The interface provided by EcPriceTable to the SDSRV to update product prices.
2. The interface provided by MsAcUsrProfileMgr and MsUsProfile to the SDSRV to verify and update user accounts of shipped products.
3. The interfaces provided by SDSRV to the CLS CSCI to continue order processing after the product price has been computed.

4.1.12.3 Start and End Conditions

The scenario starts when a registered ECS user requests a billable data product.

The scenario is terminated when the SDSRV receives a complete acquire command.

4.1.12.3 Preconditions

1. The ECS registered user has an account established in the Billing and Accounting Application Service (BAAS) CSCI.
2. The acquire request is pending (the first ten steps of SP11 have been completed and the last three are pending completion).
3. The ECS user has enough funds to fulfill a request.

4.1.12.4 Scenario Description

1. SDSRV sends a pricing request to MSS/BAAS for estimates of each subrequest that can be priced.
2. SDSRV retrieves the science user's available balance to determine if enough funds exist to honor the request.
3. SDSRV sends the product price to the CLS CSCI to verify further processing of the order; in this scenario, the request to complete the order is given by the ECS user via the CLS CSCI.
4. SDSRV requests the MSS/BAAS to deduct the price of the current request from the ECS user's User Profile database balance.
5. SDSRV places a firm request to acquire the data.

Table 4-8 gives a detailed account of this scenario.

Table 4-8. Product Request Pricing and User Profile Update

Step#	Ename	Primitive References	Interface Client	Interface Server	Description
1	Request pricing estimates	SP 58_B	SDSRV	MSS/BAAS	Retrieve price(s) of current request(s)
2	Check available user funds	SP 59_B	SDSRV	MSS/BAAS	Retrieve ECS user's available balance from User Profile database.
3	Request verification of order processing	send SetServiceName(service:string="Estimate") to DsClCommand.	CLS	SDSRV	SDSRV requests the CLS CSCI to verify order processing by having CLS send the appropriate DsClCommand.
3	Initiate request to deduct given amount from User Profile's available balance	SP 60_B	SDSRV	MSS/BAAS	Update the User Profile database
4	Confirm product request	Acquire (last 3 steps of SP 11)	CLS	SDSRV	Complete the acquire process; notify the client.

4.1.13 FOS/SDPS Interfaces

4.1.13.1 Introduction

This section captures the interfaces used by Flight Operations Subsystem CSCI (FOS) to send data to the Science Data Server CSCI (SDSRV) and Ingest CSCI (INGST). To illustrate the interoperability of these three subsystems, a series of scenarios (similar in format and content to the scenarios found in section 4), is presented.

FOS sends the following categories of data to the SDSRV:

1. Real-time Telemetry
2. Ground Telemetry
3. Events
4. Configuration Data
5. Operational Data
6. Plans and Schedules
7. orbit data

Refer to FOS Database Design and Database Schema [311-CD-001-003], Appendix A, FOS database matrix for actual the description of the actual data types in the given categories above.

This section has been included in this internal interface ICD since it reuses ECS SDPS components.

4.1.13.2 Scenario 1 - SDSRV Interaction

4.1.13.2.1 Purpose

The purpose of this scenario is to demonstrate how FOS sends data pertaining any of the above FOS data categories to the SDSRV CSCI for storage and requests a search from the SDSRV CSCI for information pertaining to the information listed above or performs an acquire for data pertaining to the information listed above. This scenario also illustrates the following CSCI interfaces:

1. The use of the SDSRV for standard insert, query, and acquire methods.

4.1.13.2.2 Start and End Conditions

The scenario starts by FOS having information for insertion into the SDSRV.

The scenario is terminated by either FOS receiving query results or email containing the reference need to retrieve stored data.

4.1.13.2.3 Scenario Description

1. FOS submits an insert request to the SDSRV.
2. FOS can either submit a query the SDSRV or perform an ACQUIRE.

Table 4-9 gives a detailed account of this scenario.

Table 4-9. Scenario 1 FOS/SDPS Interface

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Insert	SP 12	FOS	SDSRV	Insert data pertaining to the FOS data listed above into the SDSRV.
2a	Query	SP 9	FOS	SDSRV	Query the SDSRV for information needed by FOS.
2b	Acquire	SP 11	FOS	SDSRV	Submit a request to distribute requested data to FOS.

4.1.13.3 Scenario 2 - Advertisement of FOS Data

4.1.13.3.1 Purpose

The purpose of this scenario is to demonstrate how the PLANG CSCI uses SDSRV to acquire Data Availability Schedules (DAS) which are provided to SDSRV by FOS. This scenario illustrates the following CSCI interfaces:

The use of FOS DMS and the SDSRV CSCI.

4.1.13.3.2 Start and End Conditions

The scenario starts upon an event that triggers SDSRV to notify ADSRV.

The scenario is terminated by the PLANG receiving the UR to retrieve a particular DAS.

4.1.13.3.3 Scenario Description

1. PLANG searches for a particular service or advertisement on the ADSRV pertaining to a given DAS.
2. PLANG submits a subscription to SDSRV for a particular DAS.
3. SDSRV notifies ADSRV that a particular event has occurred, in this instance a series of DAS are pending insertion.
4. FOS DMS sends the DAS, which happens to be the DAS needed by PLANG, to SDSRV for insertion.
5. SDSRV sends PLANG a subscription notification.
6. PLANG sends SDSRV an ACQUIRE with UR.

Table 4-10 gives a detailed account of this scenario.

Table 4-10 Scenario 2 FOS/SDPS Interface

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Search for Advertisement	SP 48_B	PLANG	ADSRV	PLANG searches for advertisements pertaining to data availability schedules.
2	Submit Subscription	SP 7	PLANG	SDSRV	PLANG submits a subscription for a DAS that will be made available in the future.
3	Advertise Event	SP 61_B	SDSRV	ADSRV	Trigger an event.
4	Insert	SP 12	FOS DMS	SDSRV	FOS DMS inserts a series of DAS into the SDSRV in which one of the DAS is the one requested by PLANG.
5	Notification	SP 8	PLANG	SDSRV	SDSRV notifies PLANG of the availability of the requested DAS.
6	Acquire with UR	SP 56_B	PLANG	SDSRV	PLANG goes through the process of acquiring the DAS.

4.1.13.3 Scenario 3 - FOS-INGST Interaction

4.1.13.3.1 Purpose

The purpose of this scenario is to demonstrate how FOS uses the INGST CSCI to insert AM-1 Science Data, or any the of data items listed in this subsection’s introduction, and provide it to the SDSRV. FOS later retrieves data from the SDSRV. This scenario illustrates the following CSCI interfaces:

1. The use of the INGST CSCI to insert data from FOS.

4.1.13.3.2 Start and End Conditions

The scenario starts upon an event that triggers SDSRV to notify ADSRV.

The scenario is terminated by the FOS receiving the UR to retrieve a particular AM-1 data.

4.1.13.3.3 Scenario Description

1. SDSRV advertises an event to the ADSRV, in this case the use of a particular flight plan to collect AM-1 data.
2. FOS DMS submits a subscription to SDSRV so that FOS can later retrieve inserted data.
3. INGEST sends AM-1 data to SDSRV for insertion.
4. SDSRV sends FOS DMS a subscription notification that the subscribed data has been inserted. This notification contains a UR that FOS DMS will need to use for retrieval.
5. FOS DMS sends SDSRV an ACQUIRE with UR command to retrieve inserted data.

Table 4-11 gives a detailed account of this scenario.

Table 4-11. Scenario 3 FOS/SDPS Interface

Step #	Ename	Primitive References	Interface Client	Interface Server	Description
1	Advertise Event	SP 61_B	SDSRV	ADSRV	Trigger and event.
2	Submit Subscription	SP 7	FOS DMS	SDSRV	FOS submits a subscription for data that it will or has inserted into the SDPS segment of ECS.
3	Insert	SP 12	INGST	SDSRV	INGST inserts AM-1 data into the SDSRV.
4	Notification	SP 8	FOS DMS	SDSRV	SDSRV notifies FOS that the requested data has entered storage.
5	Acquire with UR	SP 56_B	FOS DMS	SDSRV	FOS acquires inserted data from the SDSRV.

4.2 Interface Scenario Primitives

This section contains the Release B scenario primitives. For each scenario primitive, descriptive text accompanies an event trace. The event traces show the signature and the calling sequence of the public services being called by another ECS object.

The scenario primitives shown here, in many cases, make use of the key mechanisms described in section 5. A forward reference identifying a key mechanism will be identified in the descriptive text describing each scenario primitive.

There are more scenario primitives than are used by the system-level scenarios (see section 4.1). These primitives are required for specific interactions between the CSCI's but are not called out by one of the system-level scenarios. Primitives 48-60 have been added in support of Release B. Each Release B primitive name ends with "_B".

Note: Scenario primitive numbers are not consecutive. Missing scenario primitives were either turned into key mechanisms (see section 6) or were found to be redundant and thus covered by one of the remaining scenario primitives.

These scenario primitives define only the cross-CSCI interfaces. As such, the primitives do not expound upon the internal operation of a CSCI. Internal CSCI functionality is defined in the respective CSCI design documentation.

Table 4-4. Scenario Primitive Table (1 of 2)

	Primitive Name	Description	Loc
2	L0 Check	Ingest client validates L0 metadata	4.2.1
3a	Archive Store	SDSRV - STMGT primitive (part of SDSRV Insert - SP 12)	4.2.2
3b	Archive Retrieve	SDSRV-STMGT primitive (part of Acquire Data - SP 11)	4.2.3
4	Create Staging File	Allocate disk, [make directory], write data, [deallocate]	4.2.4
5a	Create Electronic Distribution	Distribute granules on staging file for ftp	4.2.5
5b	Create Physical Distribution	Distribute granules on hard-copy media (tape, cdrom)	4.2.6
7	Submit Subscription	Submit subscription to SDSRV/ADSRV for notification of new granules	4.2.7
8	Subscription Notification	Notify subscribers that a data product is available	4.2.8
9	SDSRV Query	Client requests ESDT selection operation (spatial search, sub-select,...)	4.2.9
10	*Advertising*	*This has been replaced by SP 48_B to reflect the current design.*	*4.2.10*
11	Acquire Data	Submit request to distribute ESDT to client WS disk or distribution media	4.2.11
12	SDSRV Insert	Submit request to push data into SDSRV - Create UR and save	4.2.12
16	V0 Inventory Query	V0 Client Inventory Query	4.2.13
17a	V0 Gateway Integrated Browse	V0 Client Browse	4.2.14
17b	V0 Gateway FTP Browse	V0 Client Browse	4.2.15
18	V0 Acquire	V0 Client Acquire	4.2.16
19	V0 Directory Searches	V0 Directory Search processed by Gateway Subsystem at a single site.	4.2.17
20	Acquire Document	COTS - DDSRV will be HTML server w/ separate file system	4.2.18
25	Activate Plan	Operations requests PLANG to activate a plan	4.2.19
26	Release DPR Job	PLANG releases DPR Job to Job Schedule	4.2.20

Table 4-4. Scenario Primitive Table (2 of 2)

	Primitive Name	Description	Loc
29	Browse	Real time retrieval of browse images - no distribution request needed	4.2.21
45	Ingest Gateway I/F	INGST creates session with ingest gateway (DAN)	4.2.22
46	Insert Document	PLANG stores plan in DDSRV	4.2.23
47	Search Document	COTS-DDSRV will be HTML server with separate file system.	4.2.24
48_B	Retrieve Service Universal Reference (UR)	An application establishes a connection to an instance of SDSRV	4.2.25
49_B	Submit OPR	Submittal of an On-Demand Processing Request to PLANG	4.2.26
50_B	Callback	Mechanism for setting and invoking callbacks between calling and server objects.	4.2.27
51_B	DDICT Query	Data Dictionary is queried by ESST	4.2.28
52_B	DIMGR Query Submission	ESST submits a search request to DIMGR	4.2.29
54_B	DIMGR Result Retrieval	Client Retrieves data from DIMGR after DIMGR has merged results from multiple SDSRVs.	4.2.30
55_1 B	Inspect Granule Value Parameters	Mechanism for SDSRV client applications to obtain the value of the parameters of a granule.	4.2.31
55_2 B	Inspect Granule UR	Mechanism for SDSRV client applications to get the UR of a granule.	4.2.32
56_B	Acquire Data with UR	Submit request to distribute ESDT based on UR	4.2.33
57_B	Ingest-Advertising Server Interface	Mechanism for populating the Ingest data type data base with service universal references.	4.2.34
58_B	Retrieve Product Price	Mechanism for retrieving local product price information from the SMC.	4.2.35
59_B	Verify Available Funds	Mechanism for SDSRV to verify if user has enough available funds to pay for a chargeable product.	4.2.36
60_B	Update User Profile Database	Transfer product cost from user’s available funds.	4.2.37
61_B	Submit Service Advertisement	Mechanism on how to advertise a service.	4.2.38

4.2.1 L0_CHECK_SP2

This scenario shows the public interface provided by SDSRV to check L0 metadata against a descriptor for an existing ESDT. This interface is used primarily by INGST during the data pre-processing phase.

The calling object must create an instance of GlParameter for each parameter/value pair included in the metadata. The parameters are then inserted into a GlParameterList object. The L0 metadata file format and the operations used to create the necessary GlParameter objects are internal to

INGST. The calling object then creates an instance of DSClESDTDescriptor, passing the SDSRV Universal Reference (UR), and an identifier corresponding to the ESDT contained in the L0 data.

The DSClESDTDescriptor class provides a Validate method which accepts a GlParameterList and returns a GlStatus. If any parameters are invalid, The GlStatus object contains enough information to enable the calling object to determine which parameters are invalid and the reason they were flagged.

See Figure 4-1.

4.2.2 STMGT_STORE_SP3a

This scenario shows how a request is made to STMGT to store data to archives. The scenario involves two CSCIs: the requesting CSCI (the owner of the CallingObject); and STMGT, which manages archival storage.

The request is made by a call from an object of the requesting CSCI to an instance of a DsStArchive_C, which is distributed from STMGT. The calling object then calls the store operation on the archive object providing the following information:

1. unique file name
2. original file name
3. path

After control is returned to the CallingObject, it deletes the archive object.

See Figure 4-2.

4.2.3 STMGT_RETRIEVE_SP3b

This scenario shows how a request to STMGT to retrieve data from archives. The scenario involves two CSCIs: the requesting CSCI (the owner of the CallingObject); and STMGT, which manages archival storage.

The request is made by a call from an object of the requesting CSCI to an instance of a DsStArchive_C, which is distributed from STMGT. The calling object then calls the retrieve operation on the archive object, providing the following information:

1. unique file name
2. original file name
3. path
4. checksum (previously calculated by STMGT)
5. file size
6. the delivery location

as an argument to the call. After control is returned to the CallingObject, it deletes the archive object.

See Figure 4-3.

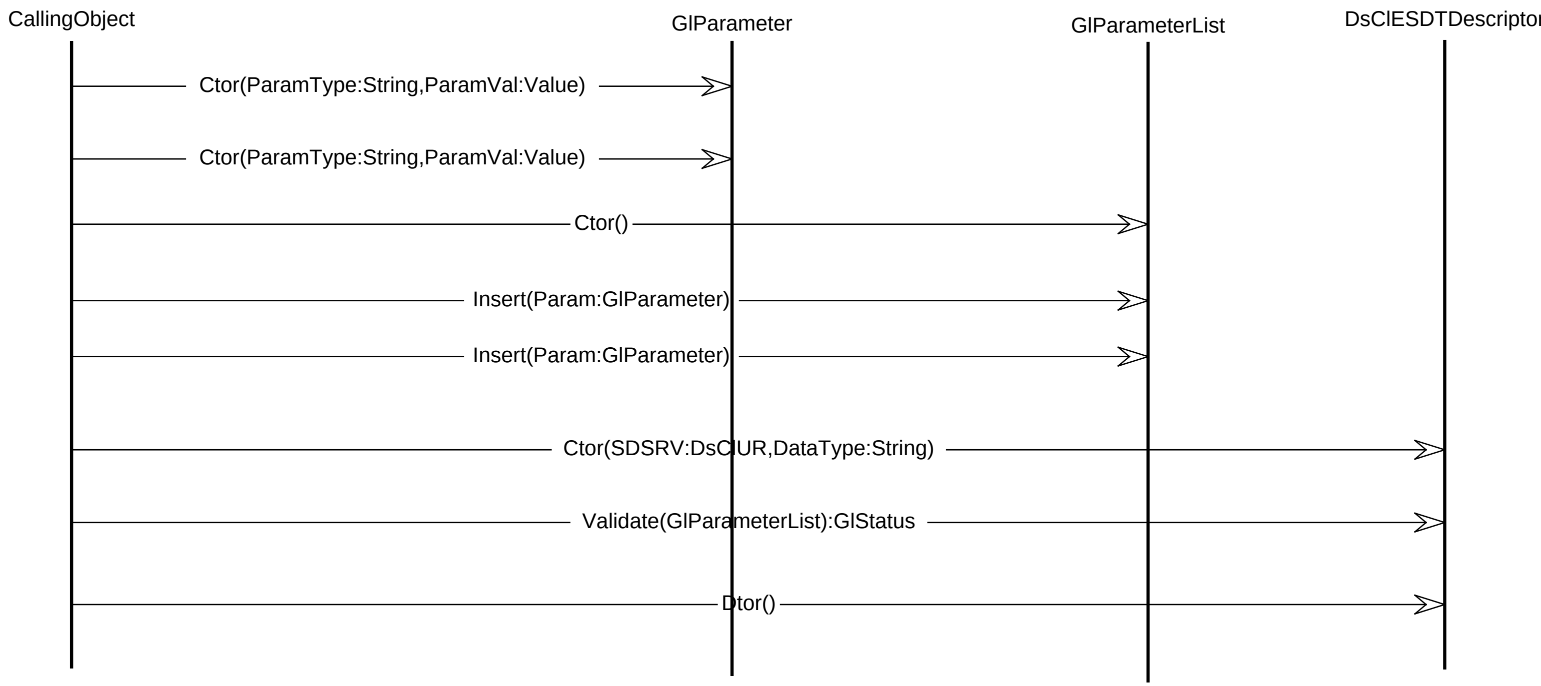

Figure 4-1. L0_CHECK_SP2 Diagram

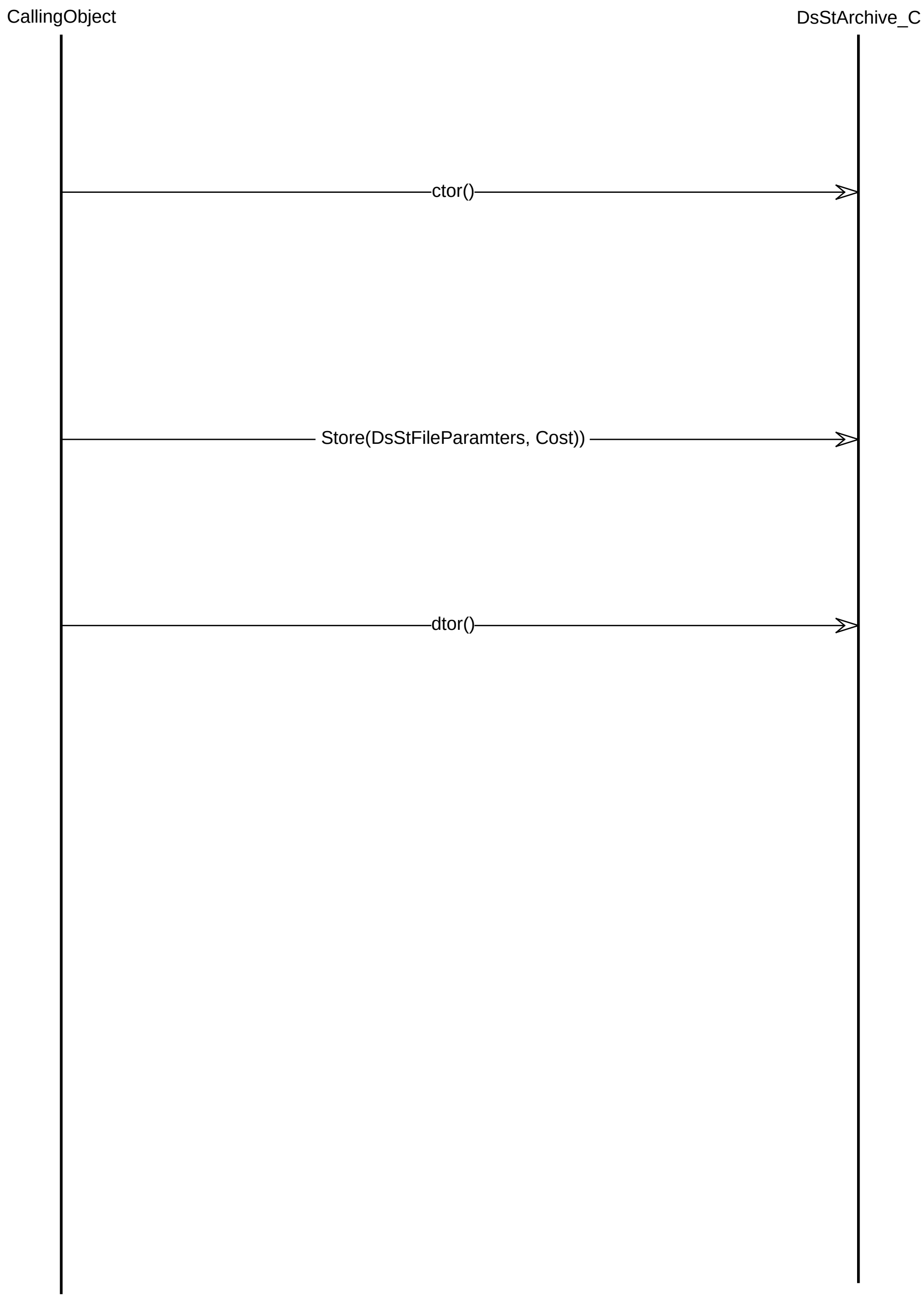

Figure 4-2. STMGT_STORE_SP3a Diagram

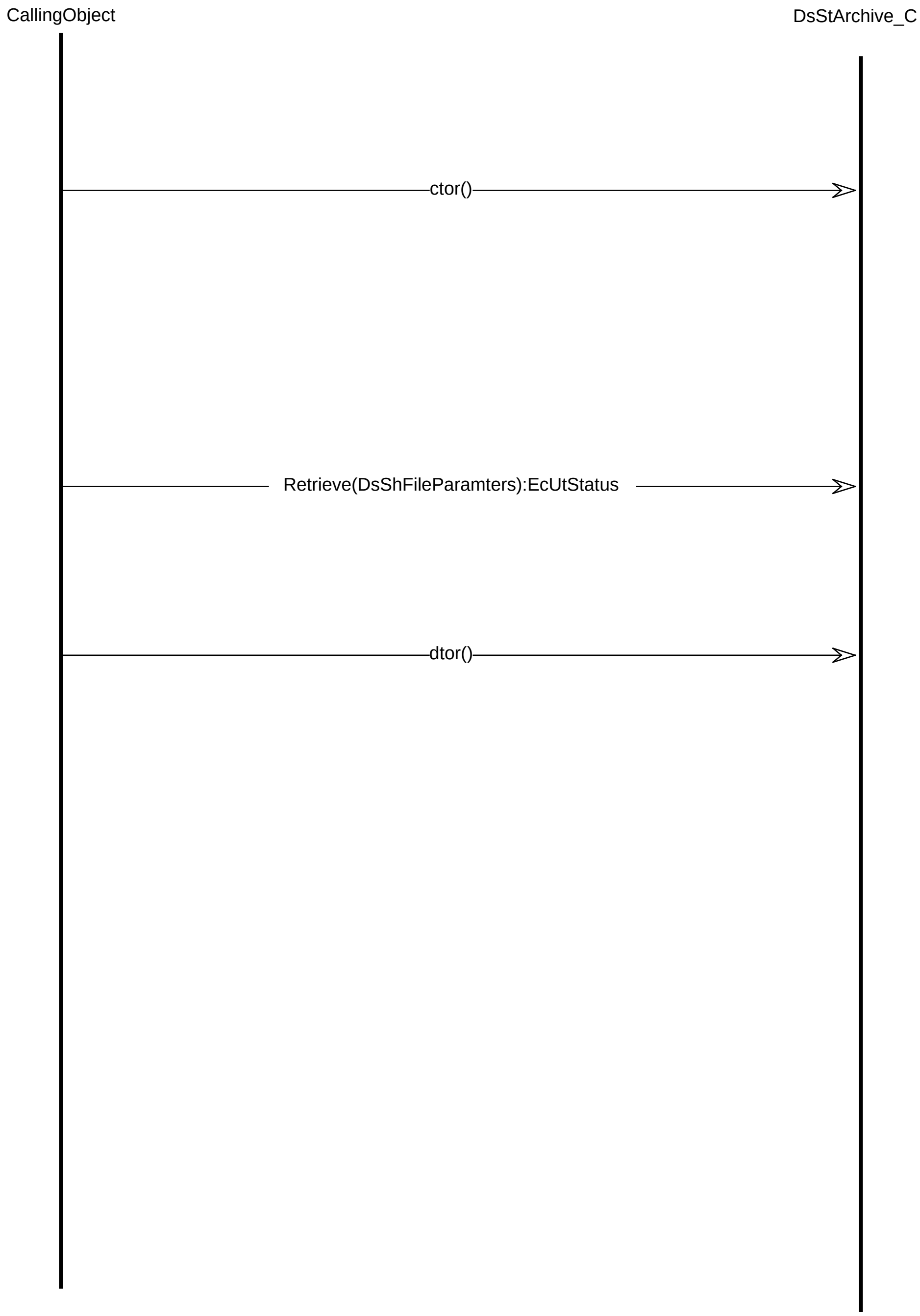

Figure 4-3. STMGT_RETRIEVE_SP3b Diagram

4.2.4 STMGT_STAGE_SP4

This scenario shows how a request is made to allocate and use a staging disk being managed by STMGT. The scenario involves two CSCIs: the requesting CSCI (the owner of the CallingObject); and STMGT, which provides the staging disk.

The request is made by a call from an object of the requesting CSCI to an instance of a DsStStagingDisk_C, which is distributed from STMGT. The allocation request is implicit in the constructor for the DsStStagingDisk object. The calling object then interacts with the DsStStagingDisk_C through method calls, which mimic direct use of the operating system's file system. In this way, directories and files can be created, moved, renamed etc. When the requesting CSCI is finished with the staging disk, the CallingObject deletes the staging disk object. The deallocation of the staging disk is implicit in the destructor for the DsStStagingDisk object.

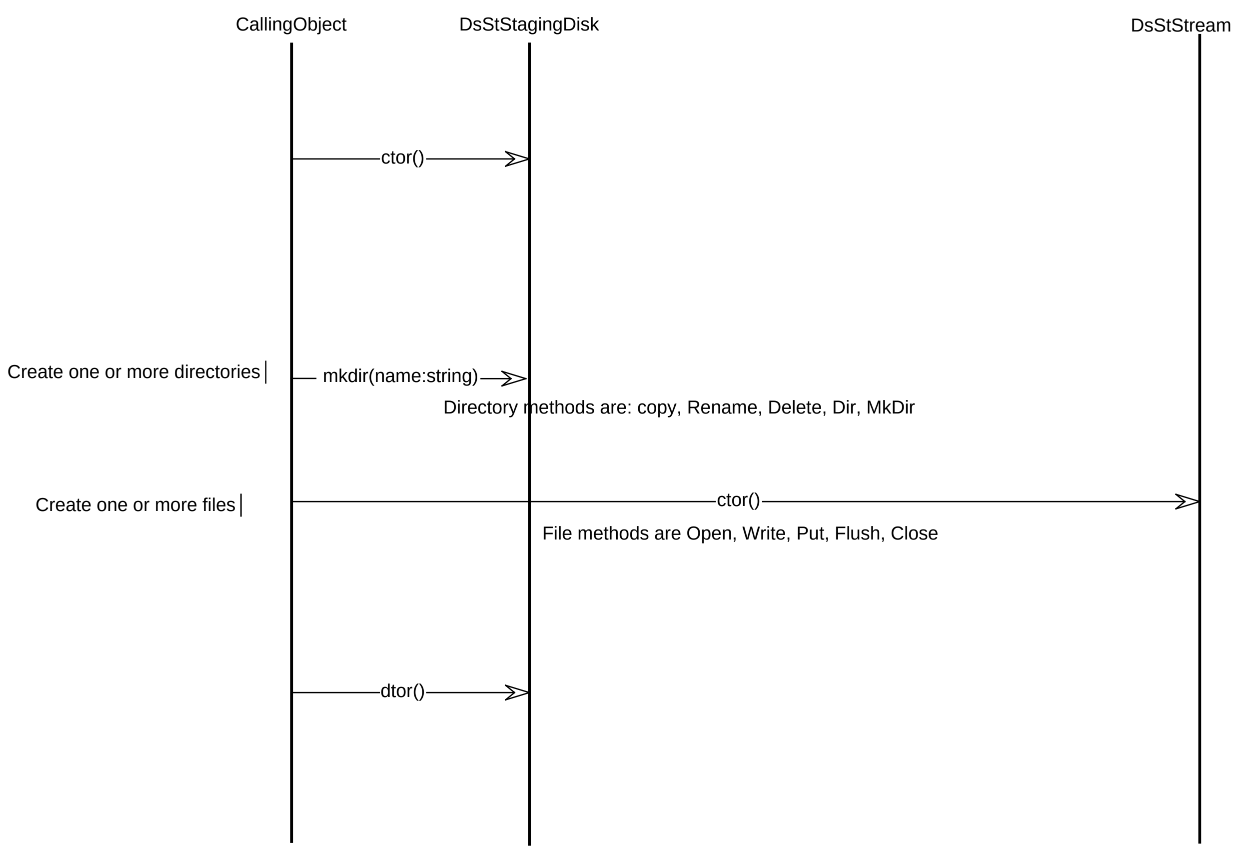

Figure 4-4. STMGT_STAGE_SP4 Diagram

4.2.5 DDIST_PULL_NOTAR_SP5a

This scenario shows how a request for electronic pull distribution of data granules is processed. The scenario involves three CSCIs: A requesting CSCI (owner of the CallingObject); DDIST, which provides the distribution service; and STMGT, which is used by DDIST in providing the distribution service.

The request is made by a call from an object of the requesting CSCI to an instance of a DsDdDistList, which is imported, but not distributed, from DDIST. DsDdDistList then initiates the creation of a number of DDIST objects which collaborate to implement the distribution service. An active object, DsDdDistRequest_S, is created on the Server side, which returns control to the requesting CSCI and then continues execution of the service on the server side. DDIST requests allocation of a network resource from STMGT using the STMGT distributed class DsStNetworkResource_C, asks the network resource to distribute the data items to the pull disk, and then deallocates the resource. DDIST notifies the requester (not the CallingObject, but rather the user on whose behalf the CallingObject made the request) via an imported utility class, GlNotification. During the processing of the request, the acceptance of the request by DDIST and the completion of the request are logged using the services of the imported utility class EcUtLogger. Since DsDdDistRequest is a distributed object, the requesting CSCI may query it for the value of its myState attribute. When DDIST is finished with the request, it sets the value of myState to indicate that the items are awaiting shipment.

See Figure 4-5.

4.2.6 DDIST_TAPE_SP5b

This scenario shows how a request for physical distribution of data granules is processed. The scenario involved three CSCIs: A requesting CSCI (the owner of CallingObject); DDIST, which provides the distribution service; and STMGT, which is used by DDIST in providing the distribution service.

The request is made by a call from an object of the requesting CSCI to an instance of a DsDdDistList, which is imported, but not distributed, from DDIST. DsDdDistList then kicks off the creation of a number of DDIST objects that collaborate to implement the distribution service. An active object, DsDdDistributionRequest_S, is created on the server side, which returns control to the requesting CSCI and then continues execution of the service on the server side. DDIST allocates a tape device from STMGT using the STMGT distributed class DsStTape, and uses a shell command to create an archive file and save the file to tape. DDIST notifies the requester (not the CallingObject, but rather the user on whose behalf the CallingObject made the request) via an imported utility class, GlNotification. During the processing of the request, the acceptance of the request by DDIST and the completion of the request are logged using the services of the imported utility class EcsUtLogger. During the processing of the request, the acceptance of the request by DDIST and the completion of the request are logged using the services of the imported utility class EcsUtLogger. Since DsDdRequest is a distributed object, the requesting CSCI may query it for the value of its myState attribute. When DDIST is finished with the request, it sets the value of myState to indicate that the items are awaiting shipment.

See Figure 4-6.

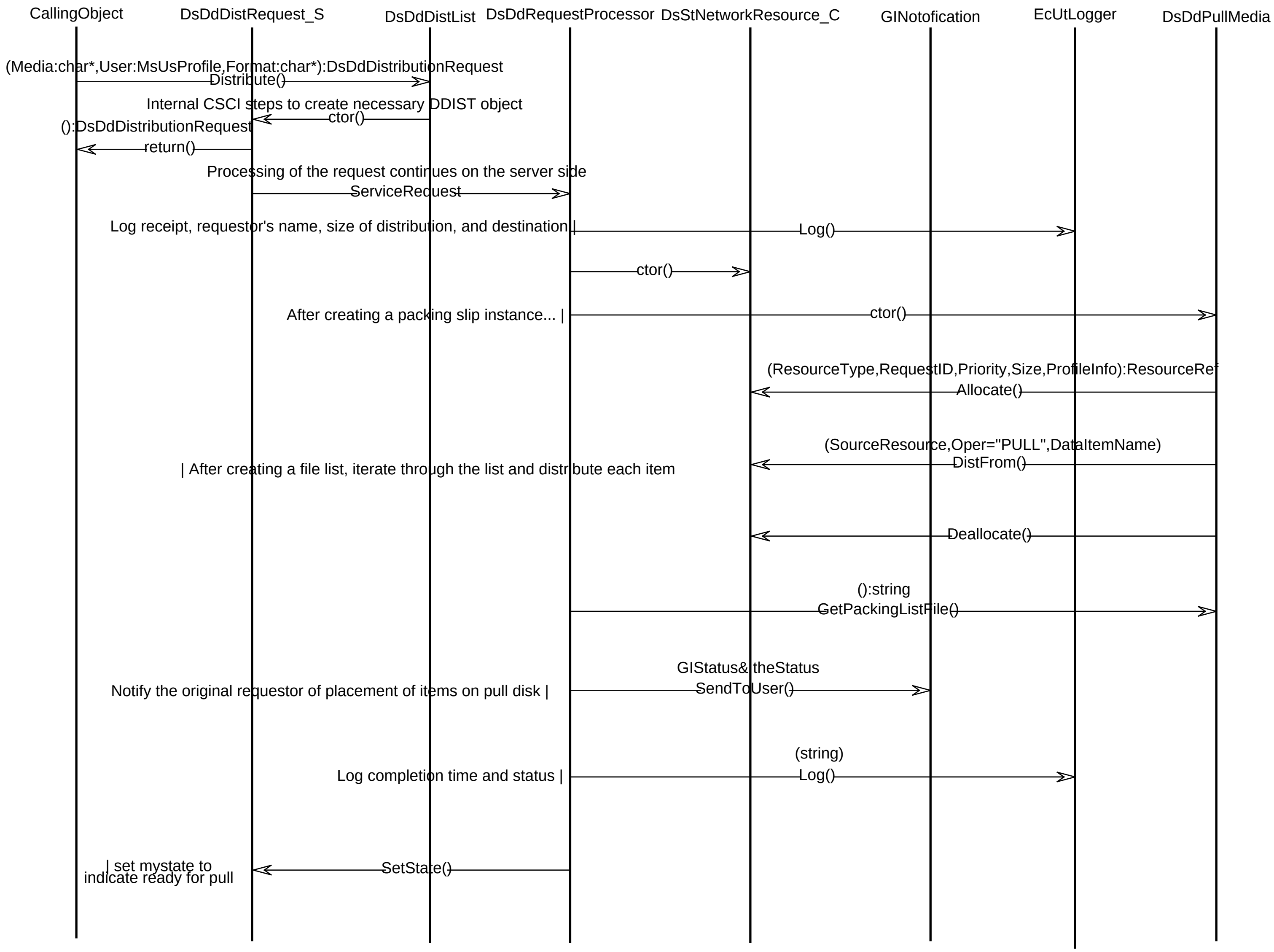

Figure 4-5. DDIST_PULL_NOTAR_SP5a Diagram

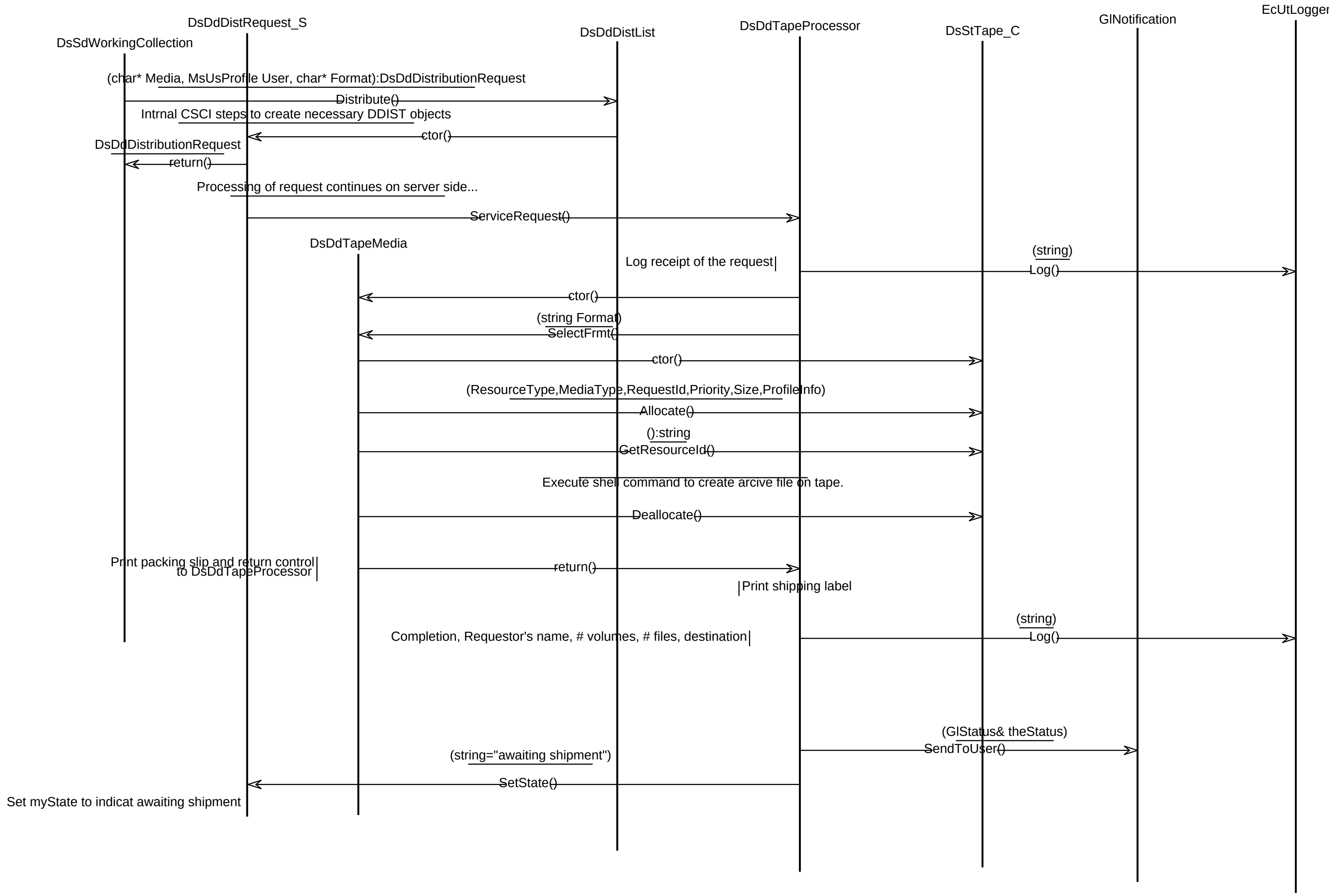

Figure 4-6. DDIST_TAPE_SP5b Diagram

4.2.7 SUBMIT_SUBSCRIPTION_SP7

This scenario primitive demonstrates the steps that the client application (CallingObject) must perform to submit a subscription on behalf of an end-user. The preconditions for this scenario primitive are that a user has selected an advertised subscription from the Advertiser and the client application has created an instance of the advertisement. The advertisement contains information about the event that will trigger the subscription. The user may specify an action for the data server to perform whenever the subscribed event occurs. For example, the user could request automatic distribution whenever new CERES02 data is arrives.

The scenario primitive starts with the client application creating a subscription, passing the advertisement (event information) and user information (notification recipient). The client then creates an action which includes a request and an optional notification text string (user-entered data). Next the DsClSubscription object's SetAction() operation is used to register the user defined action. Finally the Submit() operation activates the subscription.

See Figure 4-7.

4.2.8 SUBCRIPTION_NOTIFICATION_SP8

This scenario primitive demonstrates how ECS notifies a user who has subscribed to a service that an event has occurred related to the service. In this case, the notification is delivered to an end-user via e-mail. The scenario assumes that an event has occurred to trigger the notification (e.g., the arrival of new data).

The object that manages the subscribed service (Subscription Notifier) first creates an instance of a GlNotification object. The Subscription Notifier then invokes the SendtoUser() operation provided by GlNotification. GlNotification retrieves the user's email address from the UserProfile object (see ECS Key Mechanism 2). Next GlNotification creates an email object CsEmMailRelA provided by CSS and invokes the operations AddTo(), Subject(), AddMessage(), and Send() to build and send an email containing the appropriate notification text. The CsEmMailRelA uses the O/S mail service to deliver the notification to the user.

See Figure 4-8.

4.2.9 SDSRV_QUERY_SP9

This scenario primitive demonstrates the public interface provided by SDSRV to perform a normal search operation. The calling object constructs a query by creating an instance of DSClQuery. The query object accepts a parameter list (GlParameterList) and a callback function (GlCallback) which is invoked when the query is complete. The parameter list is a set of parameter/value pairs which are combined to define the search criteria. The search criteria is the conjunction of spatial, temporal, and keyword parameters.

The mechanics of the query execution are internal to SDSRV. The ECS callback key mechanism is used to invoke the calling object's function when the query has completed. A typical callback function in this scenario will extract the ESDT references that were inserted into the DSClESDTReferenceCollector during the query execution. This is accomplished via standard container class iterators: GetFirst(), GetNext, etc. Which return ESDT references in this case.

GlCallback is not implemented as an object but implemented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

See Figure 4-9

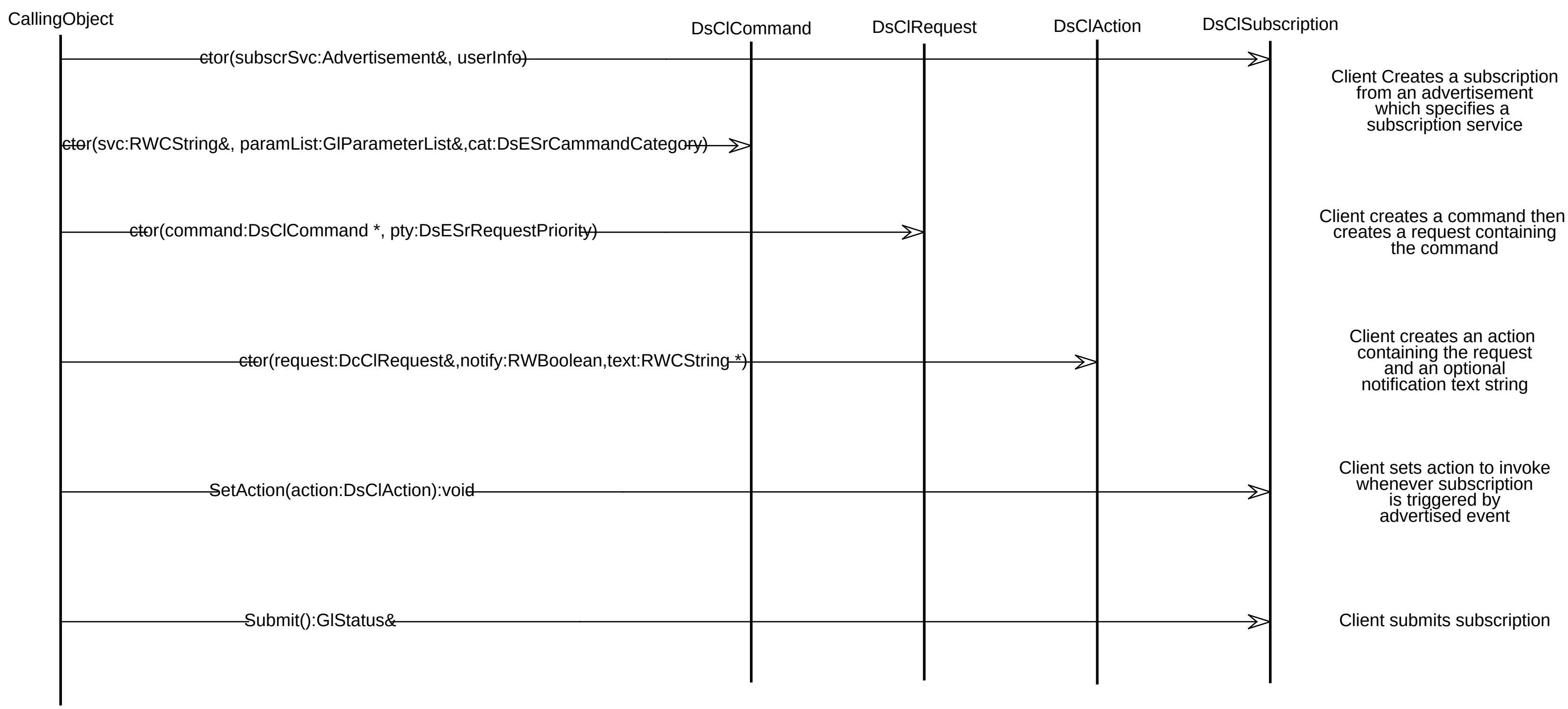

Figure 4-7. SUBMIT_SUBSCRIPTION_SP7 Diagram

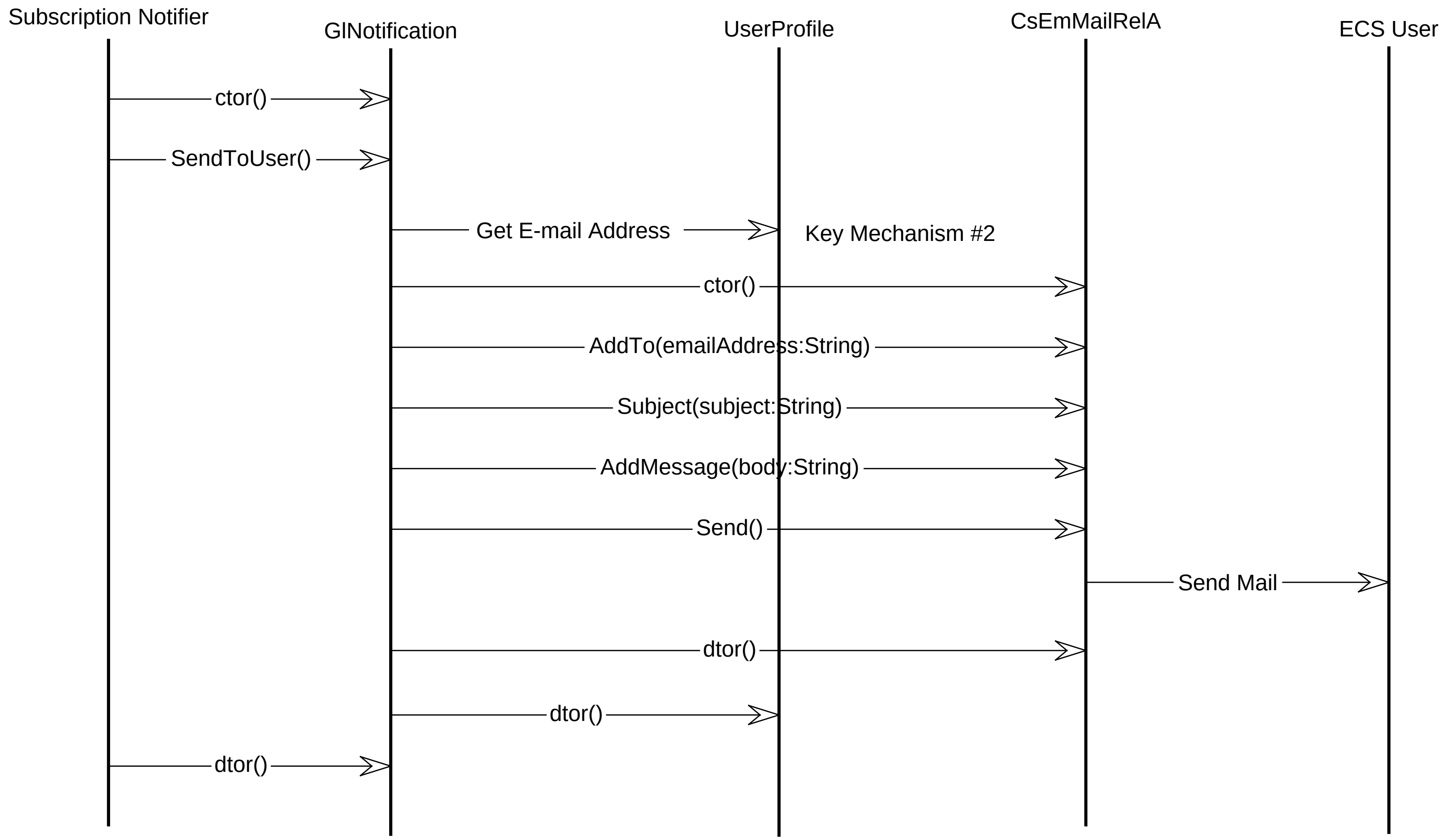

Figure 4-8. SUBSCRIPTION_NOTIFICATION_SP8 Diagram

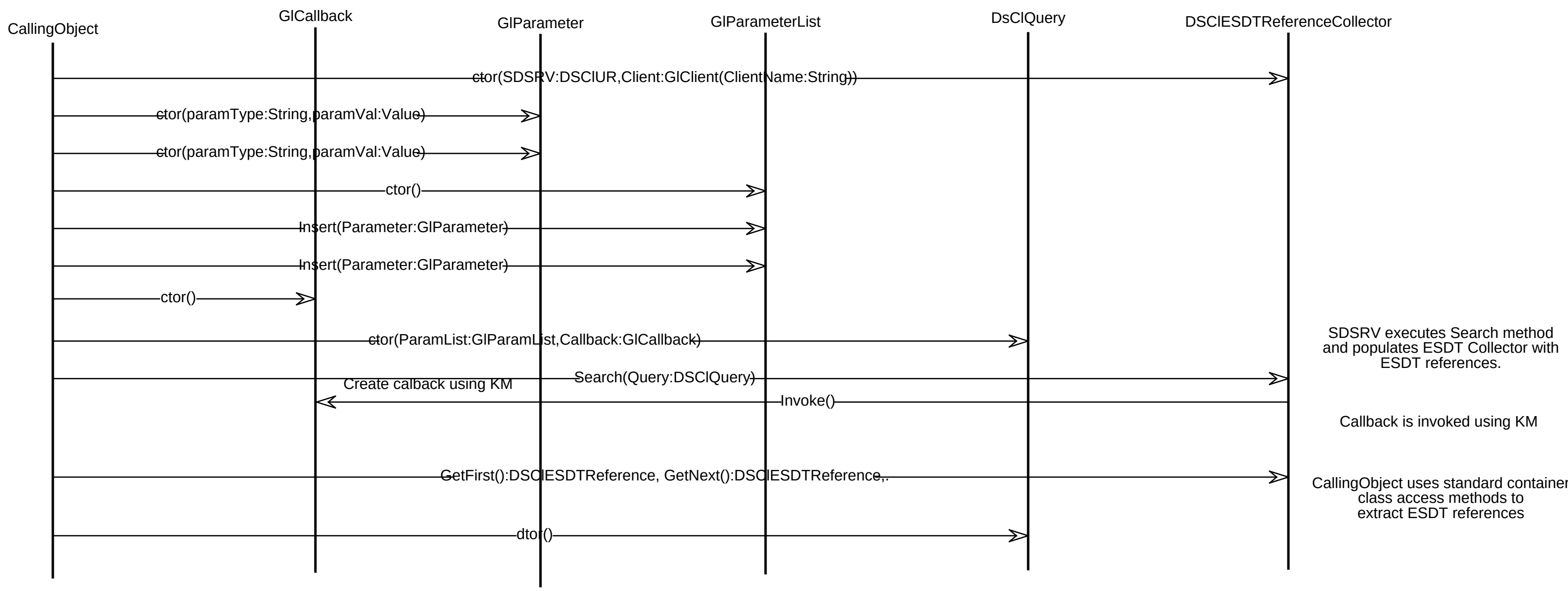

Figure 4-9. SDSRV_QUERY_SP9 Diagram

4.2.10 ADVERTISING_SP10

Originally designed for Release A, this scenario privmitive is no longer valid with the current design. The mechanism that replaces this particular scenario primitive is Retrieve Service Universal Reference (UR) SP 48_B.

4.2.11 ACQUIRE_DATA_SP11

This scenario (see Figure 4-10) shows how a request is made to acquire data from the SDSRV. The scenario involves two CSCIs: the requesting CSCI (the owner of the CallingObject); and SDSRV, which provides the data.

The main interface with SDSRV is through an instance of the SDSRV distributed class, DsClESDTReferenceCollector, which has been provided the identity of a GlCallback instance to be called upon completion of the request. An instance of DsClRequest, which is imported but not distributed, is created and one or more instances of DsClCommand are inserted into it. Into each DsClCommand object is inserted a list of parameters which define the data to be acquired, and an attribute of DsClCommand is set to indicate that it is an “Acquire” command. The calling object then asks each request to submit itself. Each DsClRequest instance then inserts itself into the DsClESDTReferenceCollector instance, which then submits the requests to SDSRV. The requesting CSCI continues after making this synchronous call. Later, SDSRV notifies the requesting CSCI of the completion status by calling the invoke operation on the specified GlCallback instance. The call to GlCallback is a local call from a distributed SDSRV object.

GlCallback is not implemented as an object but implemented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

4.2.12 INSERT_DATA_SP12

This scenario (see Figure 4-11) zshows how a request is made to insert data into SDSRV. The scenario involves two CSCIs: a requesting CSCI (the owner of the CallingObject); and SDSRV, which accepts the insert request. The purpose of the scenario is to detail the inter-CSCI interactions. Therefore, steps which are internal to a CSCI are summarized through textual comments on the Event Trace Diagram that corresponds to the scenario.

The main interface with SDSRV is through an instance of the SDSRV distributed class, DsClESDTReferenceCollector, which has been provided the identity of a GlCallback instance to be called upon completion of the request. An instance of DsClRequest, which is imported but not distributed, is created and one or more instances of DsClCommand are inserted into it. Into each DsClCommand object is inserted a list of parameters which define the data to be stored, and an attribute of DsClCommand is set to indicate that it is an “Insert” command. The calling object then asks each request to submit itself. Each DsClRequest instance then inserts itself into the DsClESDTReferenceCollector instance, which then submits the requests to SDSRV. The requesting CSCI continues after making this synchronous call. Later, SDSRV notifies the requesting CSCI of the completion status by calling the invoke() operation on the specified GlCallback instance. The call to GlCallback is a local call from a distributed SDSRV object.

GlCallback is not implemented as an object but implmented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

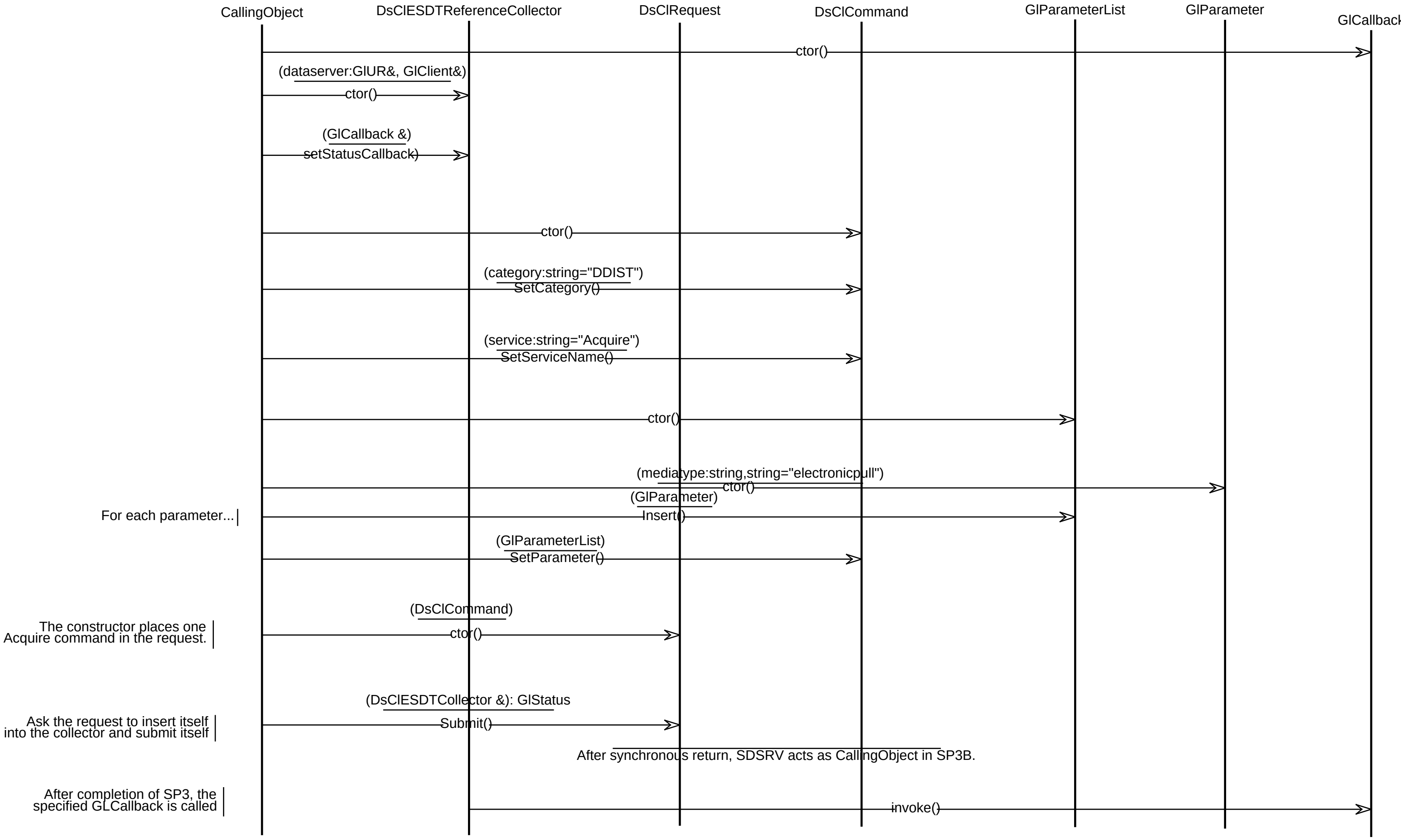

Figure 4-10. ACQUIRE_DATA_SP11 Diagram

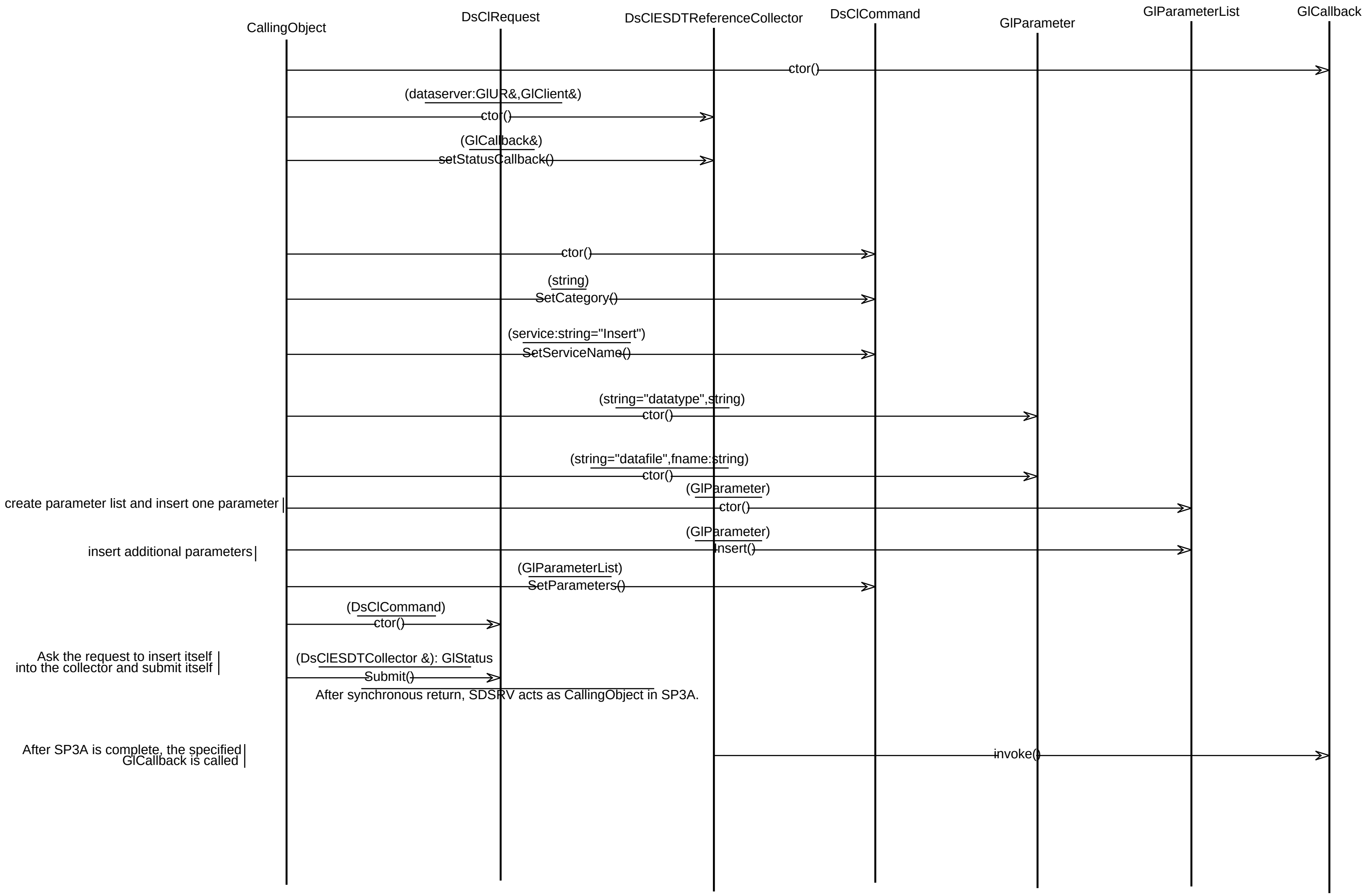

Figure 4-11. INSERT_DATA_SP12 Diagram

4.2.13 V0_GATEWAY_INVENTORY_SP16

This scenario (see Figure 4-12) describes a V0 Inventory Search Request being processed by the Gateway Subsystem at a single site. The Gateway transforms the request to the ECS protocol and does mapping of V0 attribute names and parameter names to ECS attribute names and parameter names. It then forwards the search request to the SDSRV. SDSRV processes the inventory search and returns the granule product information to Gateway. Gateway then transforms the result to V0 protocol and does mapping of ECS attribute names and parameter names to V0 attribute names and parameter names. The result information is then forwarded to the V0 client.

1. When the V0ServerFrontEnd receives a Inventory Search Request from the V0 IMS System, it constructs the DmGwInvRequests object with Inventory Request ODL Tree as the argument.
2. The V0ServerFrontEnd then invokes Submit operation of the DmGwInvRequests object.
3. DmGwInvRequests object will extract the V0 requested information from the ODL Tree.
4. The object requests the DmGwV0ECSMapper to perform translation from V0 attribute names and parameter names to the ECS correspondence.
5. The object then creates the DmGwInvQuery object which is a derived class from the proxy class DsClInvQuery imported from SDSRV.
6. DmGwInvQuery is constructed with the requested inventory search information.
7. DmGwInvRequests object then constructs the DmGwInvSearchRequest object which is a derived class from the proxy class DsClESDTReferenceCollector imported from SDSRV.
8. The object then invokes the Submit operation of the DmGwInvSearchRequest object to forward the request to the SDSRV for processing.
9. The operation InvSearchComplete of the DmGwInvSearchRequest will be then invoked if the SDSRV completes the search request successfully.
10. DmGwInvRequests object will retrieve in sequence the return granule results from the DmGwInvESDTReference object which is a derived class from the proxy class DsClESDTReference.
11. Then the object will request the DmGwV0ECSMapper to perform translation from ECS attribute names and parameter names to the V0 correspondence.
12. The ODL Tree for the search result will then be shipped by the V0ServerBackEnd to the V0 IMS System.

GlCallback is not implemented as an object but implmented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

NOTE: An ODL Tree is the data structure used by the V0 kernel library to pass ODL (Object Definition Language) across the network.

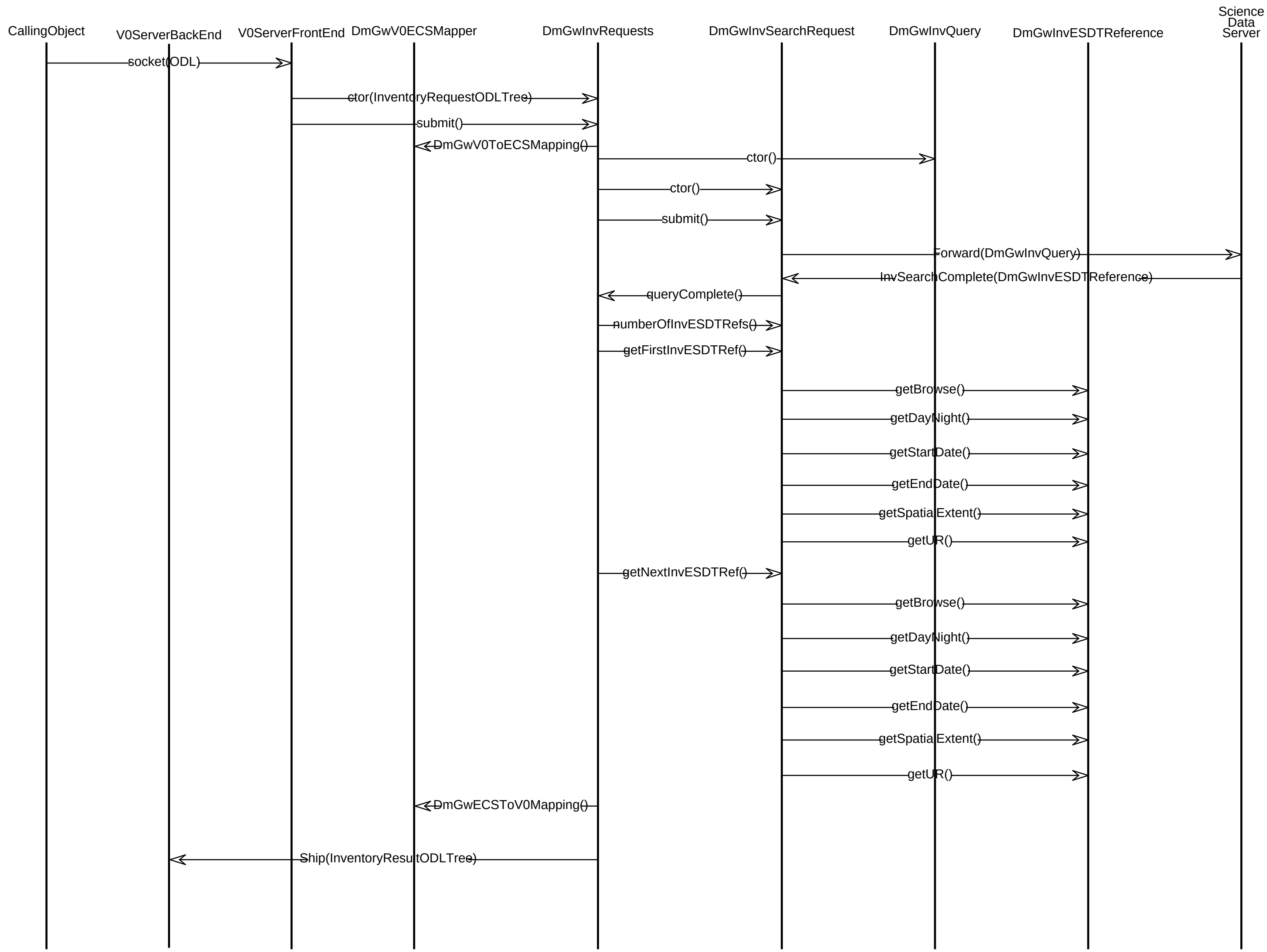

Figure 4-12. V0_GATEWAY_INVENTORY_SP16 Diagram

4.2.14 V0_GATEWAY_INTEGRATED_BROWSE_SP17a

This scenario (see Figure 4-13) describes a V0 Integrated Browse Search Request being processed by the Gateway Subsystem at a single site. The Gateway transforms the request to the ECS protocol and does mapping of V0 attribute names and parameter names to ECS attribute names and parameter names. It then forwards the search request to the SDSRV. SDSRV processes the browse search request and transmits the result information and browse image requested to the Gateway. Gateway transforms the result to V0 protocol and does mapping of ECS attribute names and parameter names to V0 attribute names and parameter names. The result information and the browse image is then forwarded to the V0 client.

1. When the V0ServerFrontEnd receives an Integrated Browse Search Request from the V0 IMS System, it constructs the DmGwV0BrowseRequest object with Browse Request ODL Tree as the argument.
2. The V0ServerFrontEnd then invokes Submit operation of the DmGwV0BrowseRequest object.
3. DmGwV0BrowseRequest object will extract the V0 requested information from the ODL Tree.
4. The object then requests the DmGwV0ECSMapper to perform translation from V0 attribute names and parameter names to the ECS correspondence.
5. The object creates the DmGwBrowseRequest object which is a derived class from the proxy class DsClRequest imported from SDSRV.
6. The object then invokes the Submit operation of the DmGwBrowseRequest to forward the request to the SDSRV for processing.
7. The operation BrowseComplete of the DmGwBrowse Request will be invoked if the SDSRV completes the search request successfully.
8. DmGwV0BrowseRequest then requests the DmGwV0ECSMapper to perform translation from ECS attribute names and parameter names to the V0 correspondence.
9. The ODL Tree for the search result and the Browse Image will then be shipped by the V0ServerBackEnd to the V0 IMS System.

NOTE: An ODL Tree is the data structure used by the V0 kernel library to pass ODL (Object Definition Language) across the network.

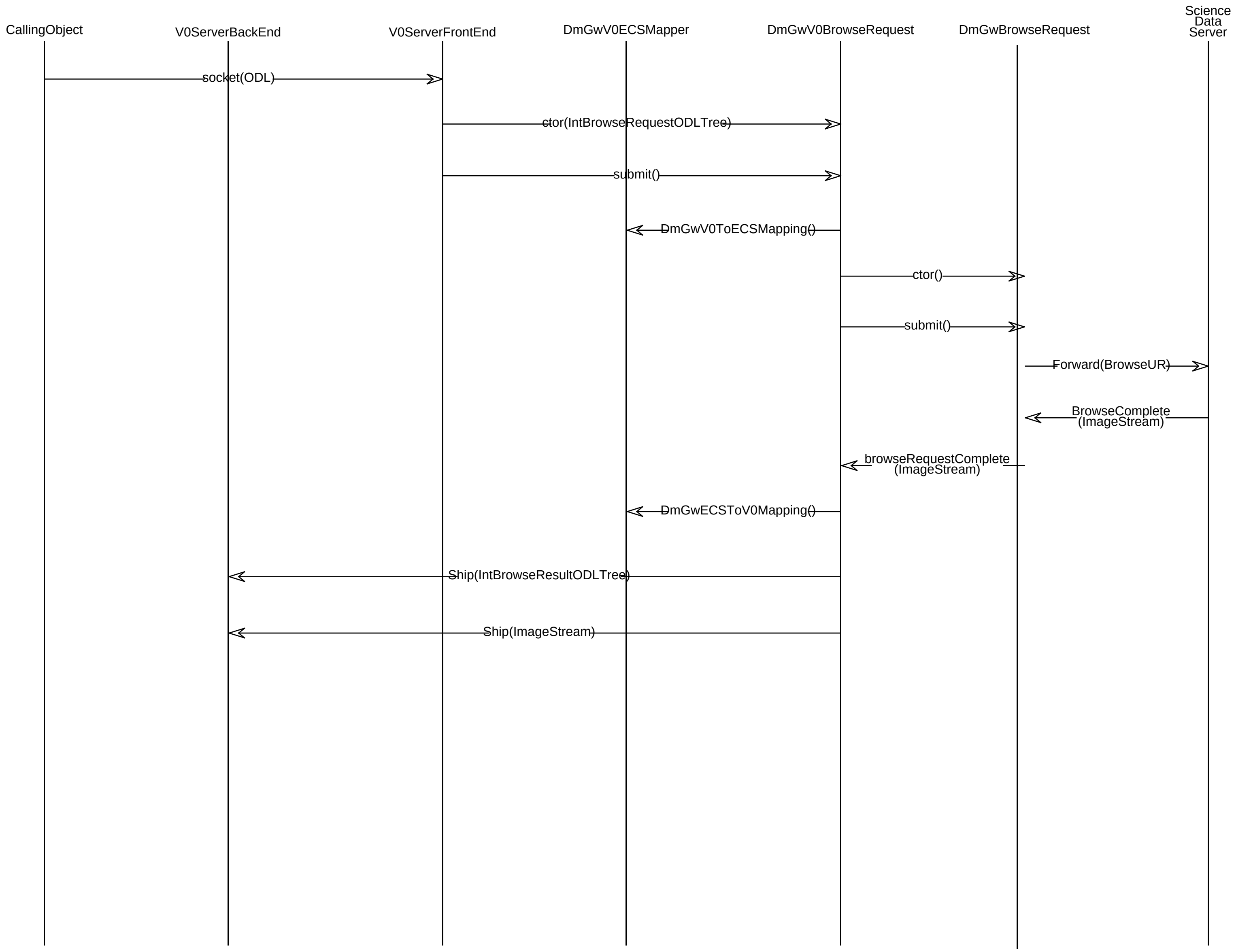

Figure 4-13. V0_GATEWAY_INTEGRATED_BROWSE_SP17a Diagram

4.2.15 V0_GATEWAY_FTP_BROWSE_SP17b

This scenario (see Figure 4-14) describes a V0 FTP Browse Search Request being processed by the Gateway Subsystem at a single site. The Gateway transforms the request to the ECS protocol and does mapping of V0 attribute names and parameter names to ECS attribute names and parameter names. It then forwards the search request to the SDSRV. SDSRV processes the browse search and sends the confirmation response back to Gateway. Gateway then transforms the confirmation response to V0 protocol and does mapping of ECS attribute names and parameter names to V0 attribute names and parameter names. The confirmation response will be forwarded to the V0 Client. SDSRV will stage the browse image product on its local disk, and sends the notification containing the location of Browse product directly to user via e-mail. The V0 User retrieves the Browse product via FTP.

1. When the V0ServerFrontEnd receives a FTP Browse Search Request from the V0 IMS System, it constructs the DmGwV0BrowseRequest object with Browse Request ODL Tree as the argument.
2. The V0ServerFrontEnd then invokes Submit operation of the DmGwV0BrowseRequest object.
3. DmGwV0BrowseRequest object will extract the V0 requested information from the ODL Tree.
4. The object then requests the DmGwV0ECSMapper to perform translation from V0 attribute names and parameter names to the ECS correspondence.
5. The object creates the DmGwAcquireRequest object which is a derived class from the proxy class DsClRequest imported from SDSRV.
6. The object then invokes the Submit operation of the AcquireRequest to forward the request to the SDSRV for processing.
7. The operation AcquireComplete of the DmGwAcquireRequest will be invoked if the SDSRV confirms the search request successfully.
8. DmGwV0BrowseRequest object requests the DmGwV0ECSMapper to perform translation from ECS attribute names and parameter names to the V0 correspondence.
9. The ODL Tree for the confirmation response will then be shipped by the V0ServerBackEnd to the V0 IMS System.
10. SDSRV will stage the Browse Image on its local disk, and sends the notification containing the location of Browse product directly to user via e-mail.
11. The V0 User retrieves the Browse Image via a FTP.

NOTE: An ODL Tree is the data structure used by the V0 kernel library to pass ODL (Object Definition Language) across the network.

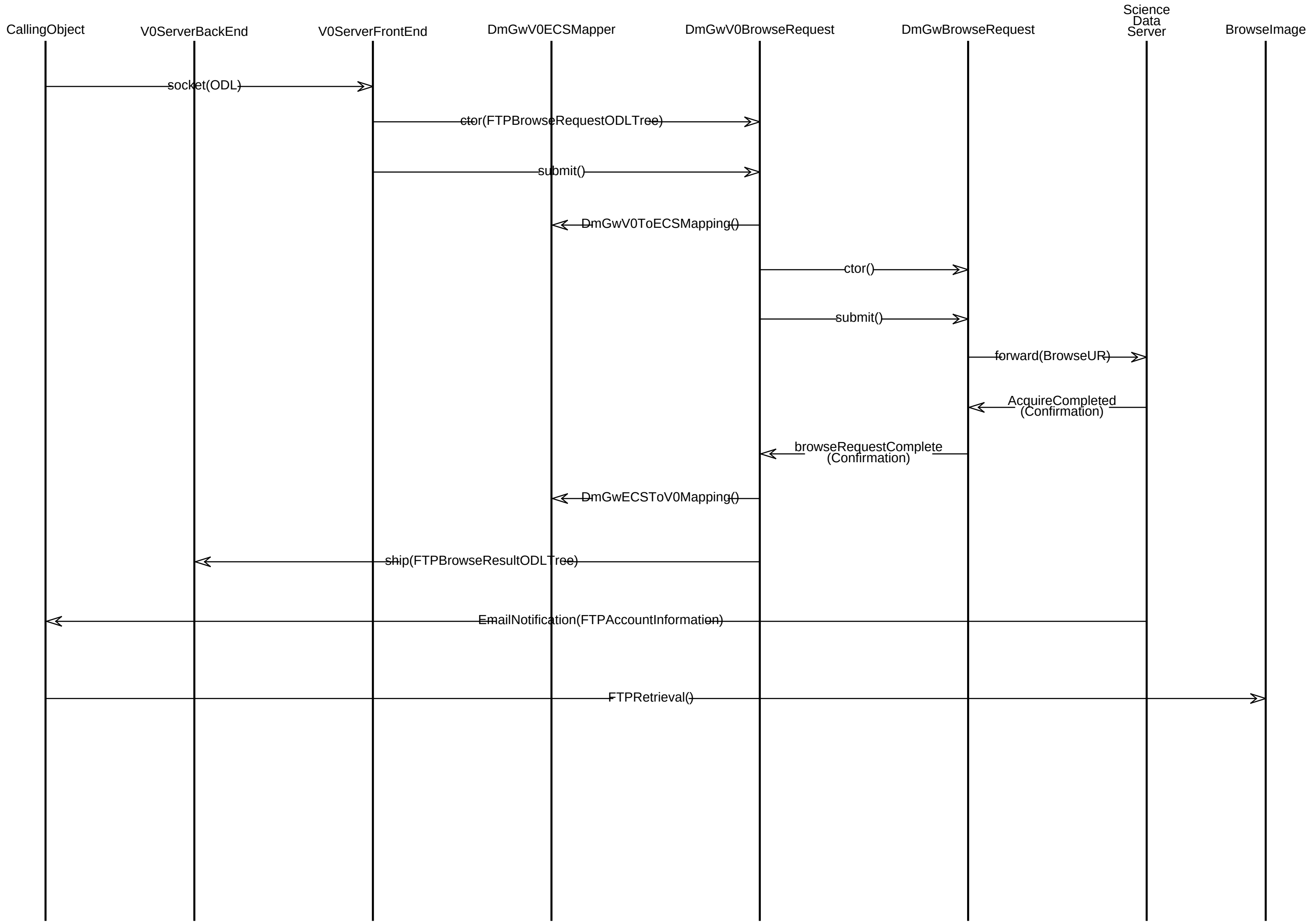

Figure 4-14. V0_GATEWAY_FTP_BROWSE_SP17b Diagram

4.2.16 V0_GATEWAY_ORDERING_SP18

This scenario (see Figure 4-15) describes a V0 Product Ordering Request being processed by the Gateway Subsystem at a single site. The Gateway transforms the request to the ECS protocol and does mapping of V0 attribute names and parameter names to ECS attribute names and parameter names. It then forwards the search request to the SDSRV. SDSRV processes the product ordering request and sends the acceptance response back to Gateway. Gateway then transforms the acceptance result to V0 protocol and does mapping of ECS attribute names and parameter names to V0 attribute names and parameter names. The acceptance response will be forwarded to the V0 client. SDSRV will then dispatch the ordered product to the Distribution Server for product distribution.

1. When the V0ServerFrontEnd receives a Product Ordering Search Request from the V0 IMS System, it constructs the DmGwProductRequest object with the Product Ordering Request ODL Tree as argument.
2. The V0ServerFrontEnd then invokes Submit operation of the DmGwProductRequest object.
3. DmGwProductRequest object extracts the V0 requested information from the ODL Tree.
4. The object then requests the DmGwV0ECSMapper to perform translation from V0 attribute names and parameter names to the ECS correspondence.
5. The object then creates the DmGwDistribution object which will contain all the media format information.
6. The object retrieves the media format information for the requested product from the DmGwDistribution object.
7. DmGwProductRequest object then constructs the DmGwAcquireRequest object which is a derived class from the proxy class DsClRequest imported from SDSRV.
8. The object then invokes the Submit operation of the DmGwAcquireRequest to forward the request to the SDSRV for processing.
9. The operation AcquireComplete of the DmGwAcquireRequest will be then invoked if the SDSRV accepts the search request successfully.
10. DmGwProductRequest will request the DmGwV0ECSMapper to perform translation from ECS attribute names and parameter names to the V0 correspondence.
11. The ODL Tree for the Acceptance Response will then be shipped by the V0ServerBackEnd to the V0 IMS System.
12. The Data Server will then dispatch the products ordered to the Distribution Server for product distribution to the V0 User.

NOTE: An ODL Tree is the data structure used by the V0 kernel library to pass ODL (Object Definition Language) across the network.

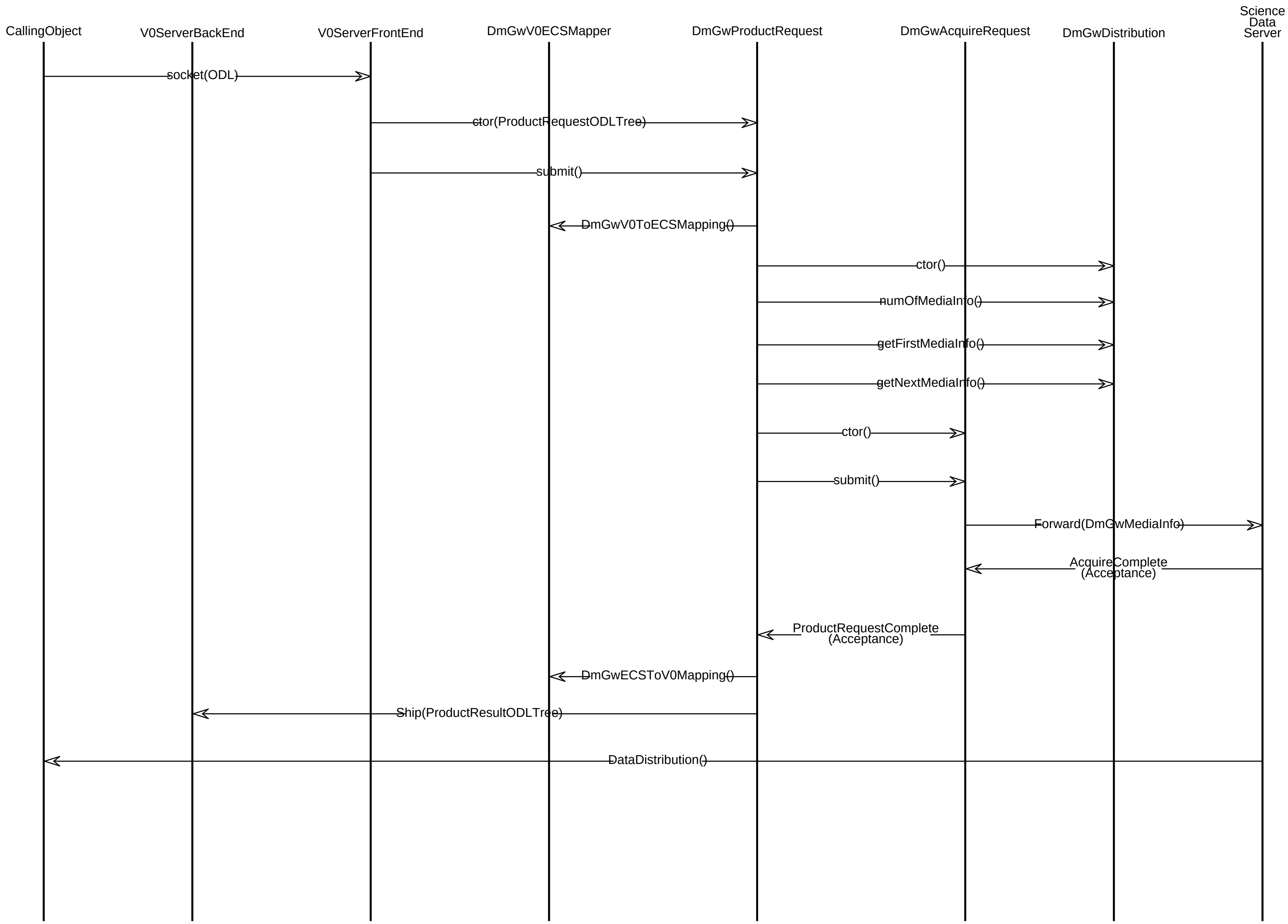

Figure 4-15. V0_GATEWAY_ORDERING_SP18 Diagram

4.2.17 V0_GATEWAY_DIRECTORY_SP19

This scenario (see Figure 4-16) describes a V0 Directory Search Request being processed by the Gateway Subsystem at a single site. The Gateway transforms the request to the ECS format and does mapping of V0 attribute names and parameter names to ECS attibute names and parameter names. It performs a search on the Gateway database to retrieve the corresponding GCMD Entry Id. The Gateway then transforms the results to the V0 protocol and does mapping of ECS attribute names and parameter names to V0 attribute names and parameter names. The directory information then is forwarded to the V0 IMS System.

1. When the V0ServerFrontEnd receives a Directory Search Request from the V0 IMS System, it constructs the DmGwDirectoryRequest object with Directory Request ODL Tree as the argument.
2. The V0ServerFrontEnd then invoke Submit operation of the DmGwDirectoryRequest.
3. DmGwDirectoryRequest object extracts the V0 request information from the ODL Tree.
4. The object will request the DmGwV0ECSMapper to perform translation from V0 attribute names and parameter names to the ECS correspondence.
5. The object then performs a directory search on the Gateway database to retrieve the corresponding GCMD Entry Id.
6. Then the object will request the DmGwV0ECSMapper to perform translation from ECS attribute names and parameter names to the V0 correspondence.
7. The ODL Tree for the search result will then be shipped by the V0ServerBackEnd to the V0 IMS System.

NOTE: An ODL Tree is the data structure used by the V0 kernel library to pass ODL (Object Definition Language) across the network.

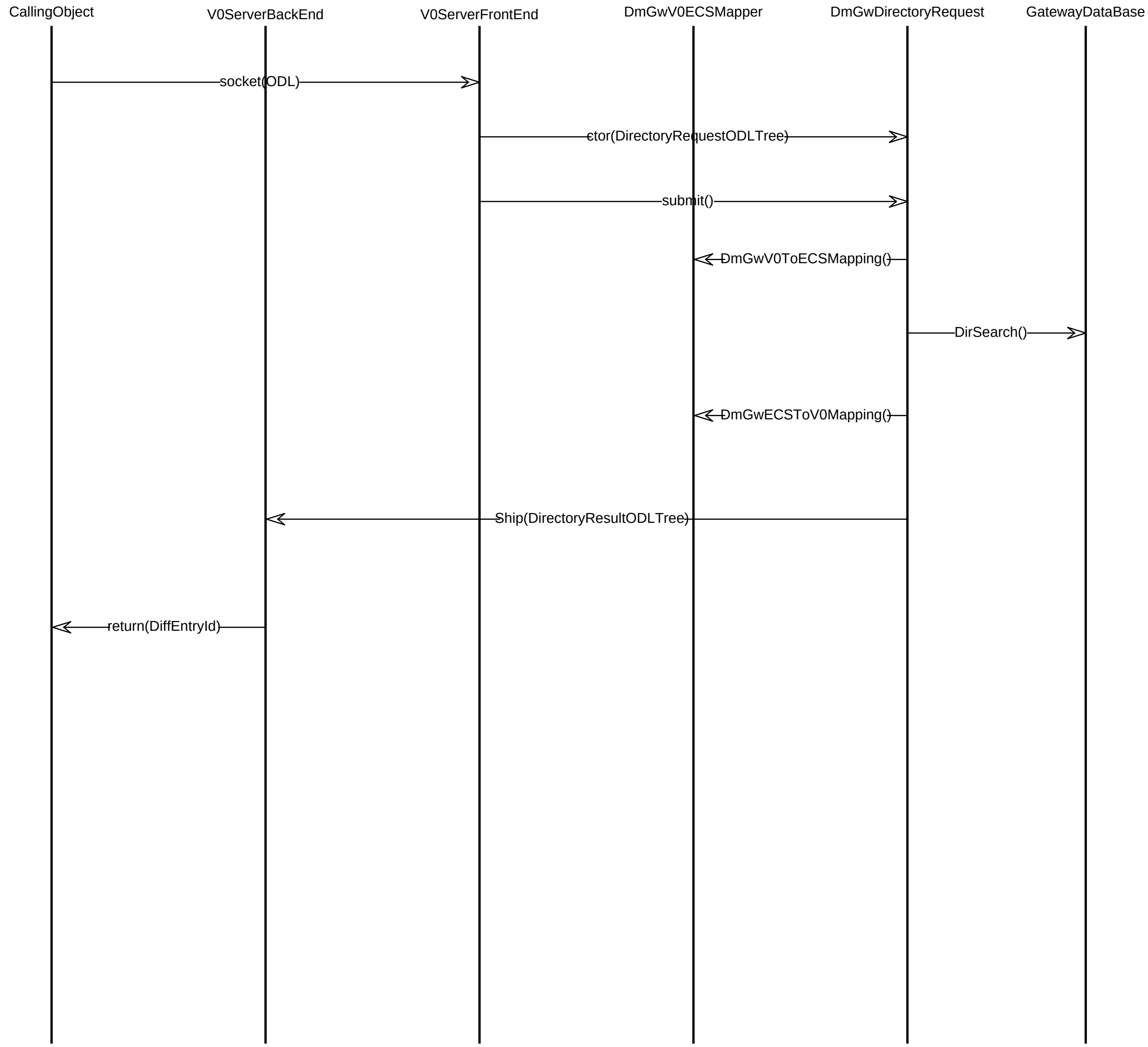

Figure 4-16. V0_GATEWAY_DiIRECTORY_SP19 Diagram

4.2.18 DDSRV_ACQUIRE_SP20

The DDSRV provides storage for its documents and document metadata, and performs the on-line distribution of documents to its HTTP clients.

In this scenario (see Figure 4-17), the client has already located the document of interest, and is requesting distribution of a Data Center Guide document via on-line HTTP connection from an interactive client capable of viewing HTML documents.

4.2.19 PLANG_ACTIVATE_PLAN_SP25

This scenario (see Figure 4-18) shows how a plan is activated within PLANG. The scenario involves two CSCIs: PLANG, which requests that the plan be activated; and PRONG, which accepts the request. It also involves Operations personnel. A Production Planner (human) selects a plan from a graphical user interface, and then activates the plan through a menu selection (ActivateSchedule). As a result of this action, PLANG undergoes a number of internal steps, culminating in a call to an instance of DpPrScheduler, which is constructed upon initialization. The call to DpPrScheduler is to create a DprJob, and a reference to the PlDpr object is passed as an argument. PRONG returns control synchronously back to PLANG and then begins to effect the plan activation.

4.2.20 PRONG_RCV_JOB_SP26

This scenario (see Figure 4-19) shows how PLANG releases a DPR to PRONG for processing, indicating that all necessary input data is available for staging. The scenario involves two CSCIs: PLANG, which releases the DPR; and PRONG, which processes it. The purpose of the scenario is to detail the inter-CSCI interactions. Therefore, steps which are internal to a CSCI are summarized through textual comments on the Event Trace Diagram that corresponds to the scenario. This scenario assumes that the PlDpr instance being released has already been submitted to PRONG by PLANG.

The interface with PRONG is through an instance of the PRONG class, DpPrScheduler, which is constructed upon initiation. PLANG asks the instance of DpPrScheduler to release a DprJob, providing a reference to the corresponding PlDpr instance as an argument to the call.

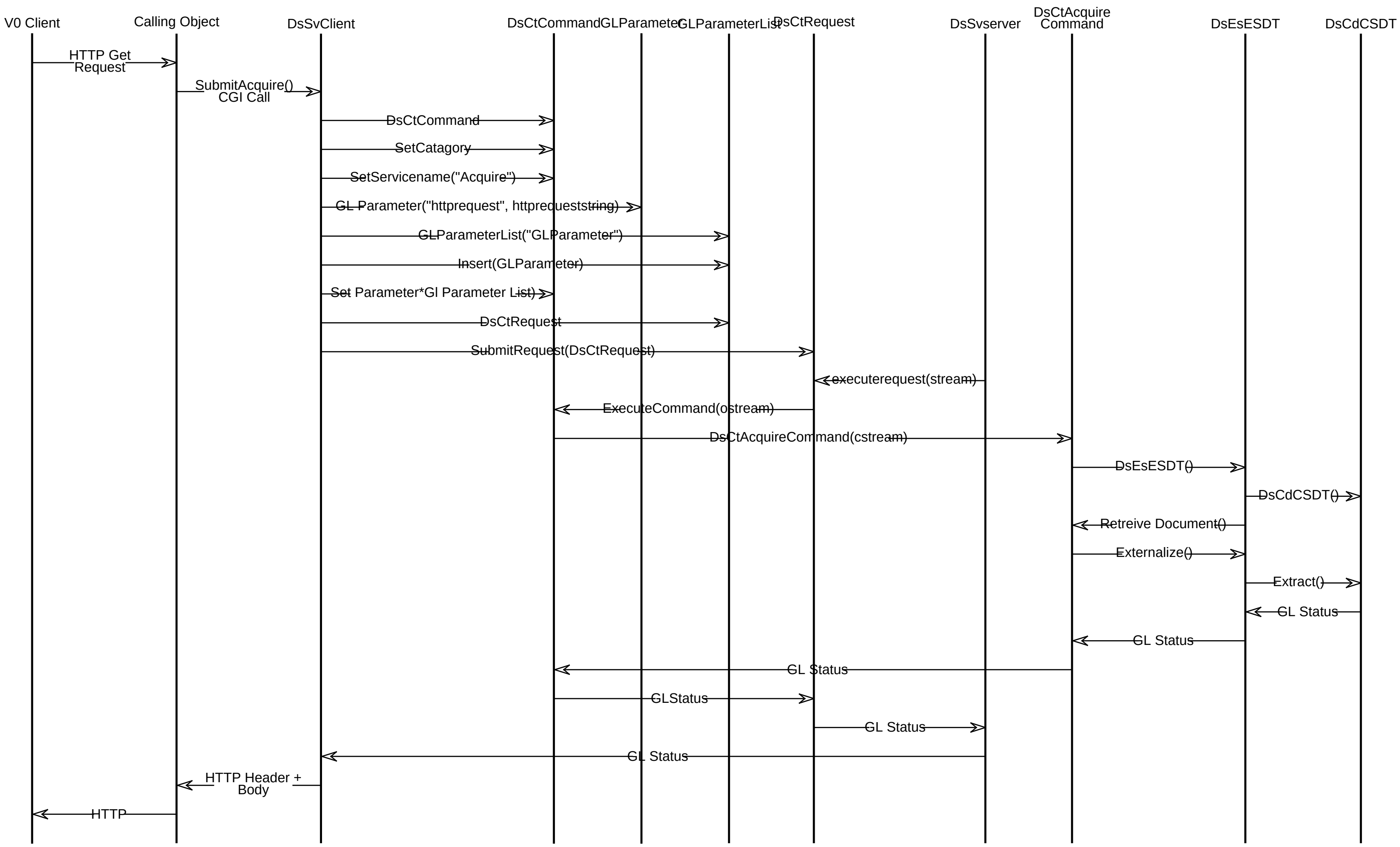

Figure 4-17. DDSRV_ACQUIRE_SP20 Diagram

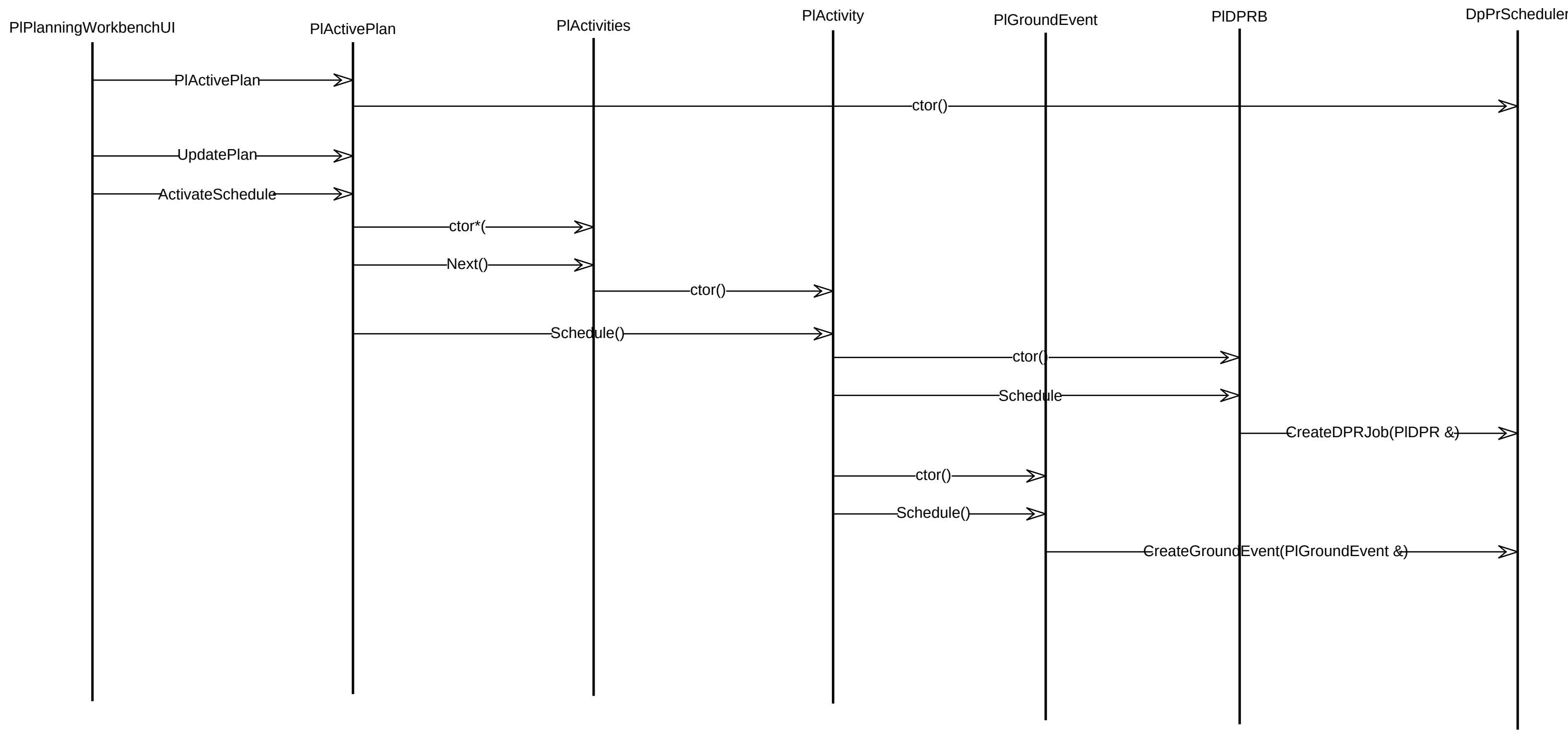

Figure 4-18. PLANG_ACTIVATE_PLAN_SP25 Diagram

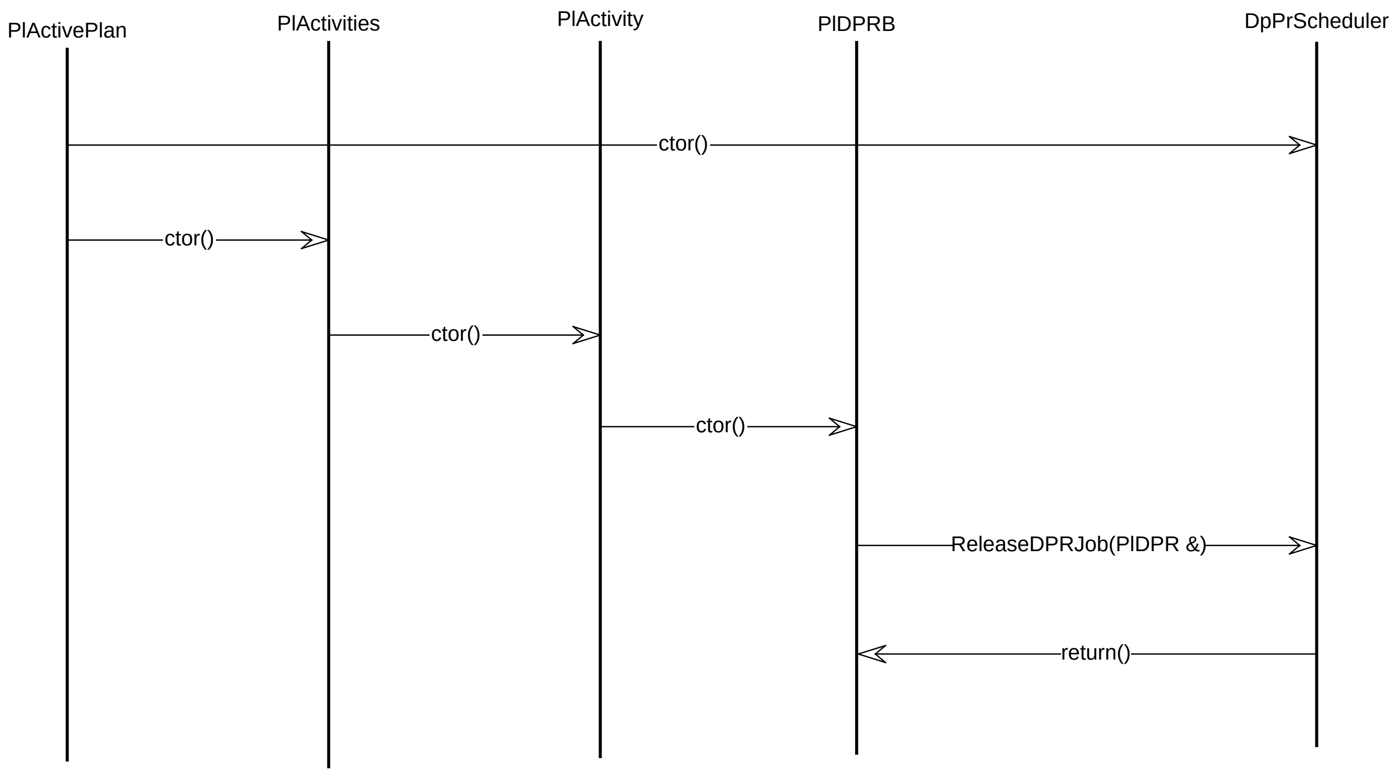

Figure 4-19. PRONG_RCV_JOB_SP26 Diagram

4.2.21 BROWSE_SP29

This scenario primitive (see Figure 4-20) demonstrates the public interface provided by SDSRV to perform a browse image retrieval. This also represents a specific use of the SDSRV's command and request interface. The scenario assumes that the user has already selected an advertisement that is linked to a particular browse image. The selected advertisement contains information about the browse image which is required by SDSRV to perform the retrieval. The calling object constructs a browse command (DsClCommand) which consumes the information provided by the advertisement.

The browse command is passed to the SDSRV in the form of a request (DsClRequest). The calling object invokes the request's SetCallback() operation to specify the callback function that will be executed when the browse image retrieval is complete. The request is started via the Submit() operation.

The mechanics of the retrieval are internal to SDSRV. The ECS callback key mechanism is used to invoke the calling object's function when the query has completed. A typical callback function in this scenario will extract the browse image by first invoking the request object's GetResults() operation. GetResults() returns an instance of GlParameterList which is derived from a standard container class. In this case, the GlParameterList is nested and contains GlParameterLists, one for each command in the request (one command in this case). Given this structure, the first at(0) returns the results of the command. An at(0) call to the nested list extracts a reference to the browse image. This is in the form of a parameter type GlBinaryP, which is a derived class of GlParameter. The parameter value() method of GlBinaryP returns a char * which contains the address of the browse image. Note that there could be several parameters in this list which contain other information about the browse image. For example, the list could also contain the UR of the ESDT from which the browse image and the size of the browse image reside.

GlCallback is not implemented as an object but implemented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

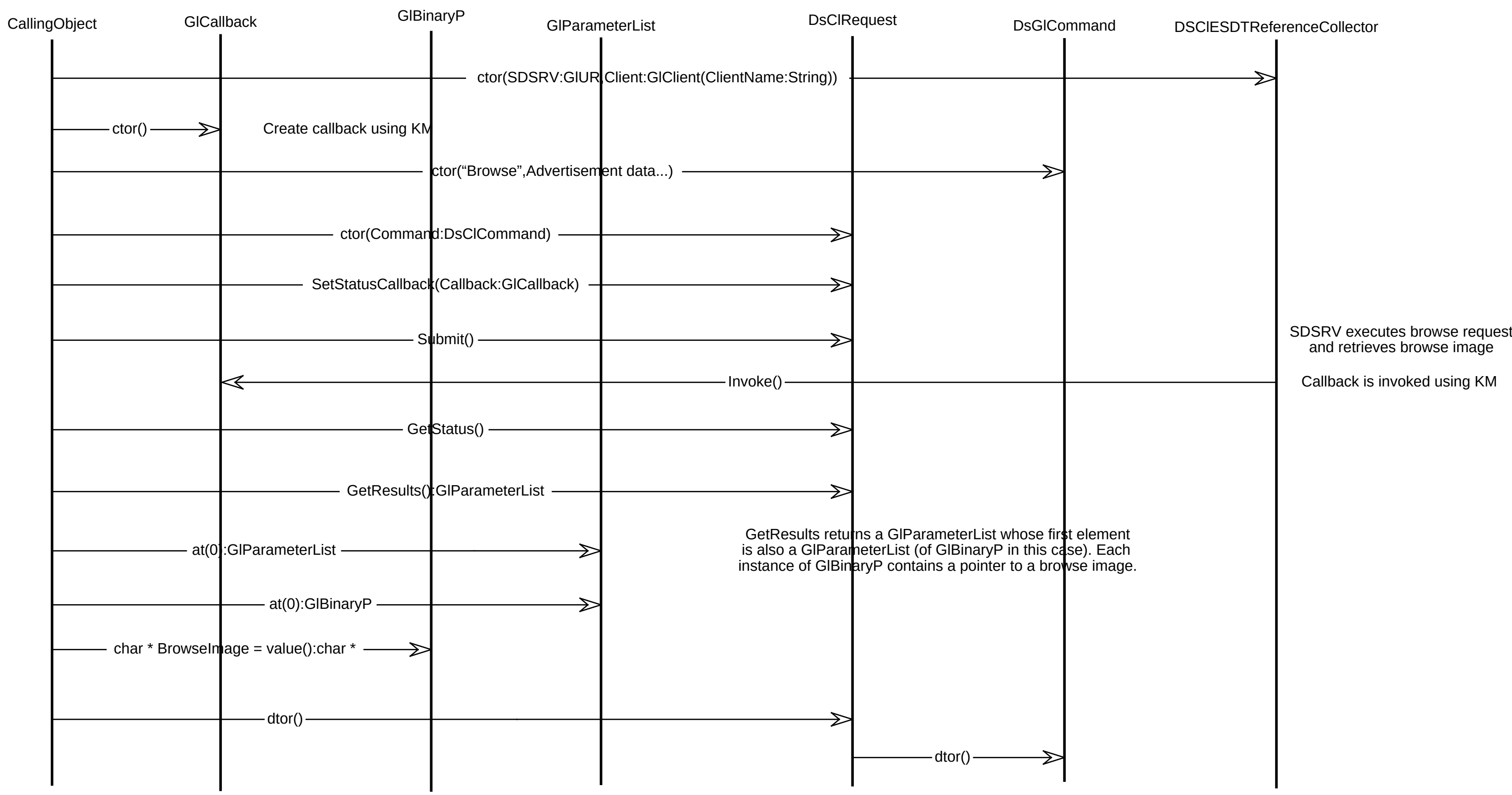

Figure 4-20. BROWSE_SP29 Diagram

4.2.22 INGEST_GATEWAY_INTERFACE_SP45

This scenario primitive (see Figure 4-21) shows the interface between The Ingest Server CSCI (INGST) and the gateway used to facilitate communications with all external data providers. The gateway communicates with external systems via TCP/IP sockets and with INGST via DCE RPCs.

The scenario primitive assumes that the InServer object has already been instantiated as part of the INGST start up procedure. In addition, the external data provider has initiated an ingest session, and completed a successful ECS login.

Once an external data provider has been authenticated, the Ingest Gateway uses its CreateSession() RPC to the Ingest Server to create an Ingest Session process. The Ingest Server then sends the InitSession() RPC to the Ingest Session process to create an instance of IngestSession.

The external data provider next sends a Data Availability Notice (DAN) to the gateway which is translated to an extDAN() RPC call to IngestSession. The Ingest Session acknowledges the DAN by sending a Data Availability Acknowledge message back to the gateway who in turn sends it on to the external data provider. The receipt of the DAN triggers a number of events internal to INGST. During this process, an instance of InResourceIF is created including the instance of an InRequestObject based on the DAN contents. InRequest creates an instance of InResourceIf which in turn creates an instance of DsStStagingDisk (see SP 4 for details) and DsStResourceProvider to allocate resources for both staging and network access. InDataTransferTask sends a message to the DsStResourceProvider copy utility to issue a CSS ftp get request.

Once successfully transferred Ingest preprocesses the data (see SP 2 for details) and sends a request to SDSRV to Archive the processed data (see SP 12 for details).

InRequest then controls the creation of DDN (data delivery notice) message which is transferred via InSession to the CSGateway. The gateway forwards this message to the external provider which sends a Data Delivery Acknowledge (DDA) message back to the gateway which forwards the DDA to the Ingest Session process via the extDDA() RPC. This causes the Ingest Session process to terminate the current session.

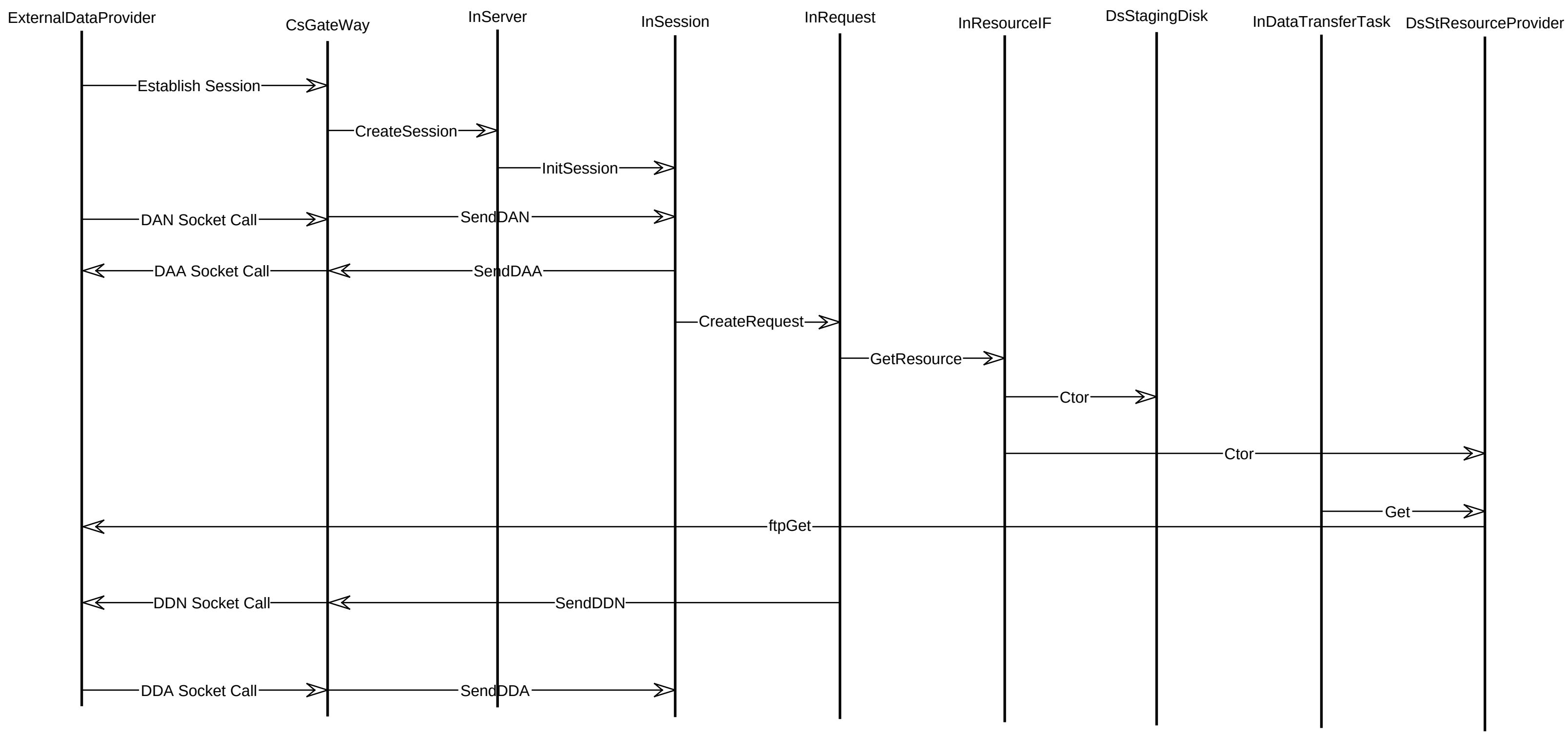

Figure 4-21. INGEST_GATEWAY_INTERFACE_SP45 Diagram

4.2.23 DDSRV_INSERT_SP46

This scenario illustrates the classes and events which interact to insert a document and its metadata into the DDSRV (see Figure 4-22). The Ingest subsystem performs metadata extraction and range checking in a manner analogous to that for science data, presenting the DDSRV with the document data file and its metadata file. The DDSRV then uses the DBMS Wrappers to index the document from its metadata file keywords. In addition, the DDSRV performs full text indexing for those document types which have a free text search defined for its ESDT. After the document has been inserted into the DDSRV a status indicating a successful insert request is returned to the Ingest Subsystem, and the document is ready for searching.

In this scenario, the DDSRV will be ingesting a Data Center Guide document (DsDoDataCenterGuide). The Ingest Subsystem has received a Data Center Guide document in HTML format, from its client interface. The Ingest Subsystem has extracted keywords from the document, performed range checking of the keywords using metadata configuration information, and has produced a parameter value list file (PVL) containing the document keywords. The Ingest Subsystem has established a connection with the DDSRV to complete the insertion of the HTML document, and its keyword and free text indexing.

4.2.24 DDSRV_SEARCH_SP47

The DDSRV provides both a free text index and keyword index on its document holdings. The keyword index uses parameter names and values consistent with the ESDT metadata stored in the SDSRV; the free text index is performed by a COTS product. Keyword integrity between the data designs of the DDSRV and the SDSRV facilitates navigation between the various layers of the data pyramid. The scenario ends with the presentation of an HTML formatted results list of references to documents which meet the search criteria. Each document reference in the results list is presented as a hyperlink in the HTML page, and points to the relevant document that is stored in the DDSRV.

In this scenario (see Figure 4-23), the client is requesting both a free text search and a keyword search for Data Center Guide documents. The client has formulated the query in WAIS from an interactive client capable of viewing HTML documents.

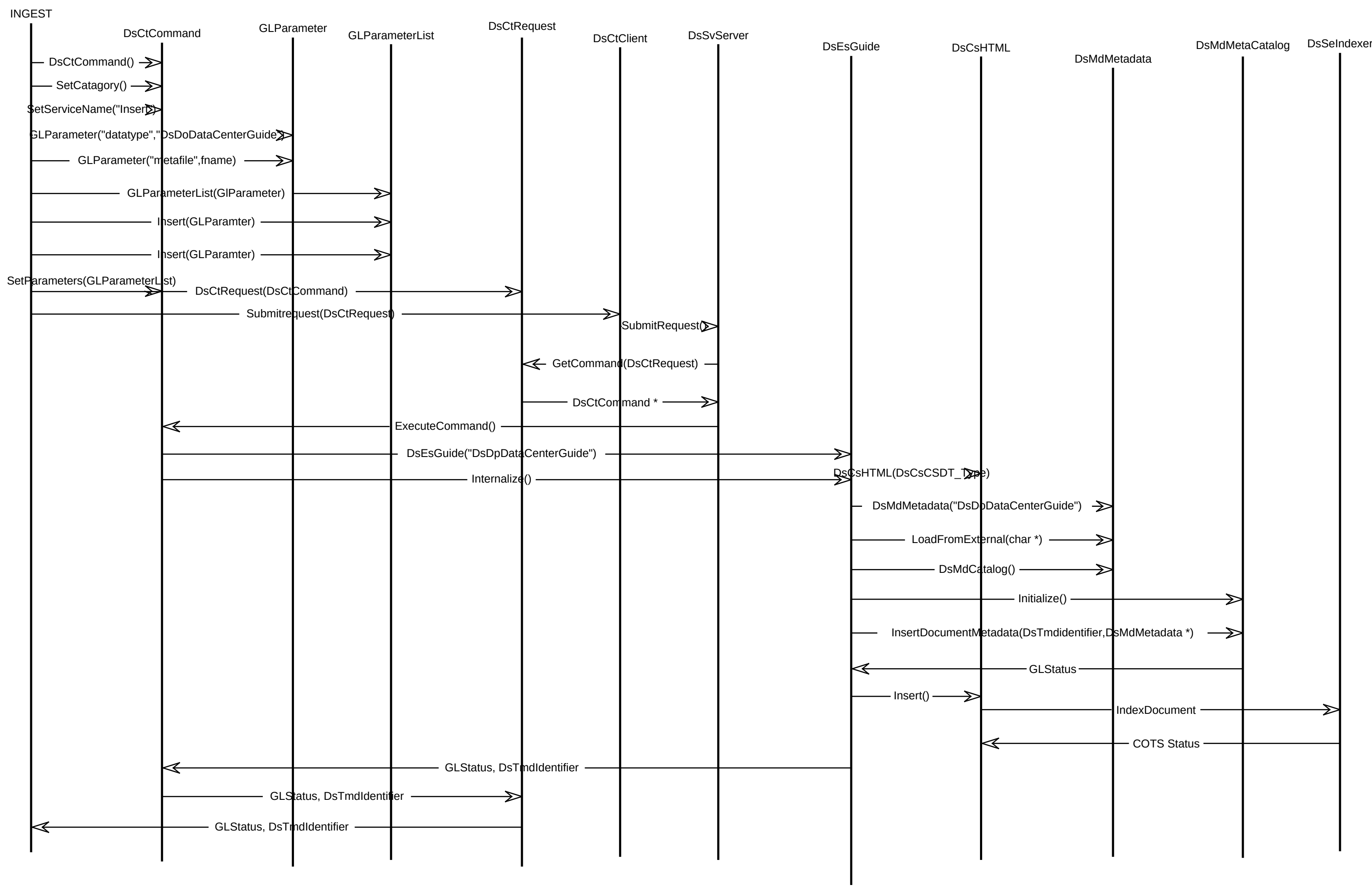

Figure 4-22. DDSRV_INSERT_SP46 Diagram

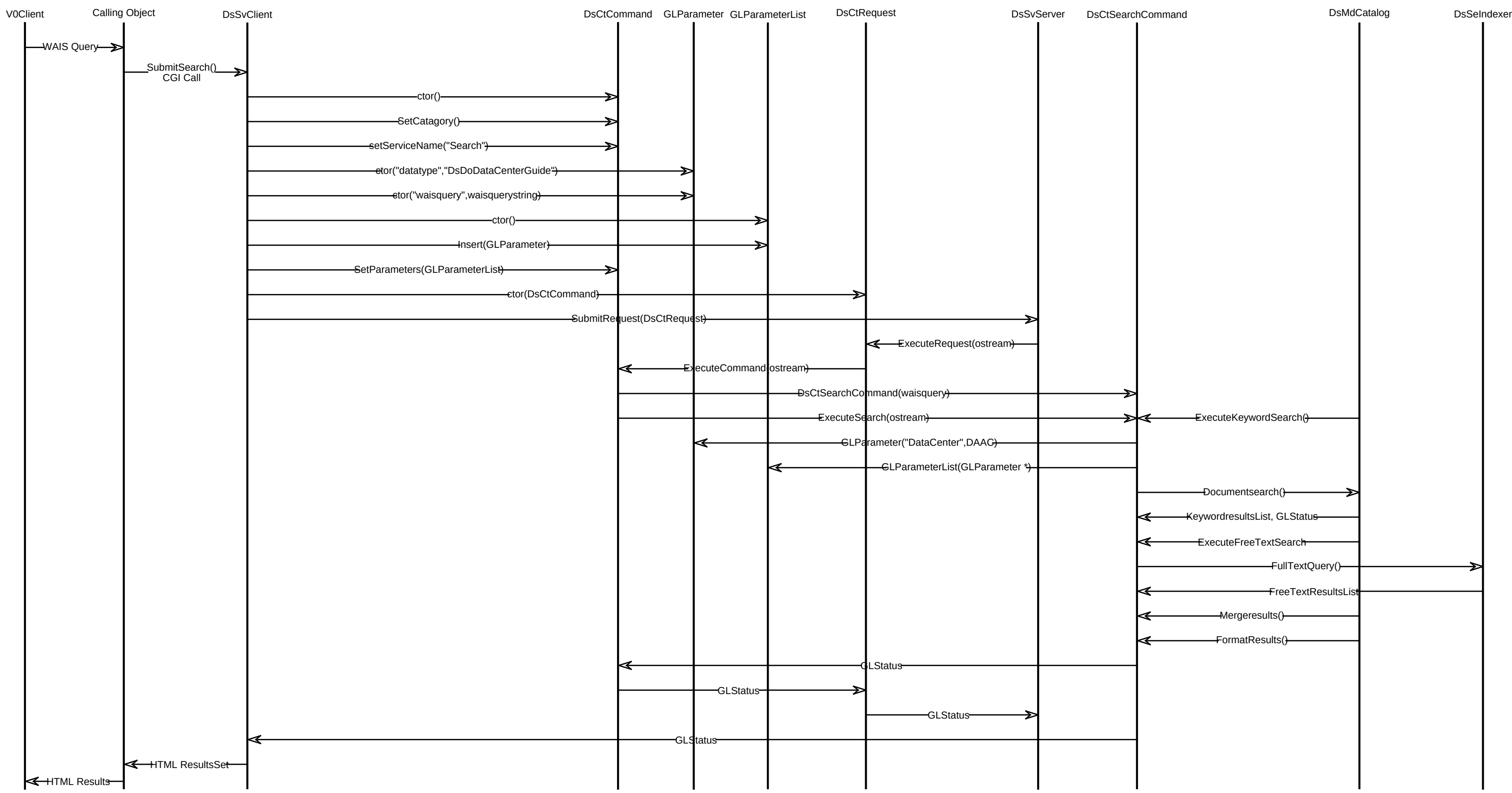

Figure 4-23. DDSRV_SEARCH_SP47 Diagram

4.2.25 RETRIEVE_SERVICE_UR_SP48_B

The event trace diagram (see Figure 4-24), illustrates how an application accesses the ADSRV to obtain the service URs for the SDSRVs involved in the information request.

The calling object (some object in the requesting CSCIs calling space) initiates a search for products by creating a IoAdProductSearchCommand object. Next, the calling object specifies the MatchType and the SearchByTitle to the IoAdProductSearchCommand object as the search criterion. For example, if the MatchType is “Exact” and the SearchByTitle is “CERES02”, then the IoAdProductSearchCommand object will return all products whose name is CERES02. The IoAdProductSearchCommand then creates a IoAdProductList object to append found Product advertisements to the current results set. Then, the calling object again calls the IoProductSearchCommand object with an operation GetResult to return the matched Product advertisements. The IoAdProduct object creates a IoAdProvider object, which retrieves the provider for the products found. The calling object then calls the IoAdProductList object with the Entries and Operator operations to iterate through the products and providers and to advance the iterator to check if a certain product is at the current DAAC. To get all the services for this product, the calling object then calls the IoAdServiceReferenceListIter to iterate through the services. While iterating, the Key operation of the IoAdServiceReferenceListIter object returns the current service ad, and the calling object then calls the IoAdService object with the GetTitle operation to check if this service ad is the Data Server Service, and its UR is then returned to the calling object.

4.2.26 SUBMIT_OPR_SP49_B

This scenario primitive (see Figure 4-25) shows how an on demand processing request (OPR) is submitted. This scenario involves three CSCIs: the calling CSCI (e.g. SDSRV); PLANG which validates the OPR; and SDSRV which searches for the input data (see the SDSRV_QUERY_SP9 scenario primitive scenario).

An OPR object is created by the calling object. The calling object initializes the appropriate instance variables when the OPR object is instantiated. The calling object asks PLANG to validate the OPR object by invoking the validate method. PLANG directs SDSRV to search for input data (see the SDSRV_QUERY_SP9 scenario primitive scenario). PLANG returns status to the calling object by initializing the appropriate instance variables accordingly (see the CALLBACK_SP50_B primitive scenario).

4.2.27 CALLBACK_SP50_B

This scenario primitive (see Figure 4-26) shows a generic mechanism for a calling object to be called back by a server object. An instance variable of a callback object is created. The calling object specifies a callback object to the server object so that the calling object can be notified of the request completion. The calling object submits the request. Sometime later, the calling object's callback is invoked upon search completion.

GlCallback is not implemented as an object but impelmented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

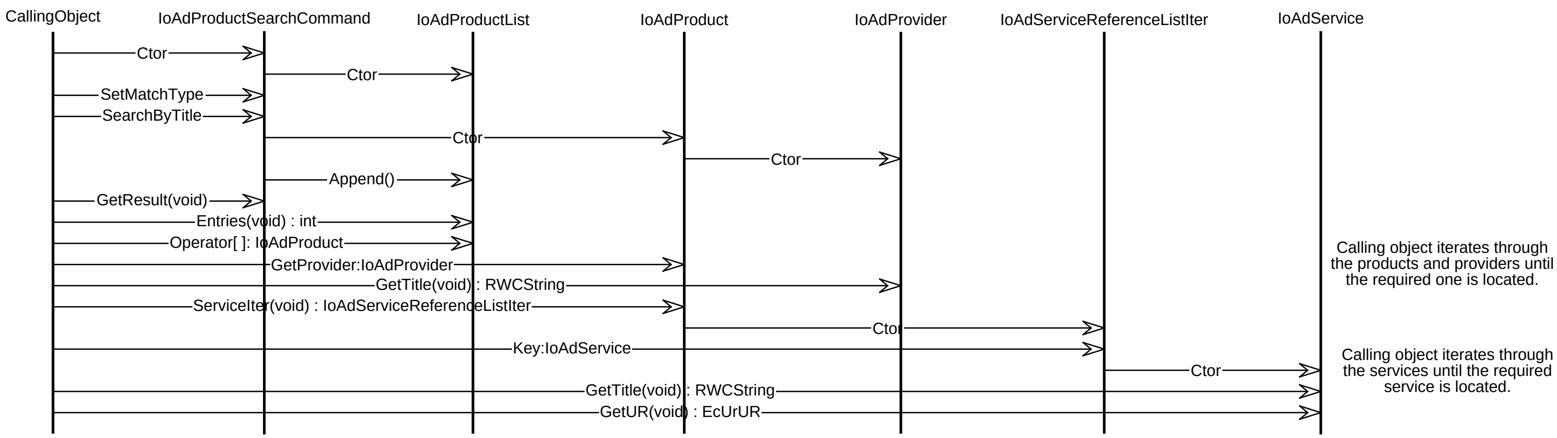

Figure 4-24. RETRIEVE_SERVICE_UR_SP48 Diagram

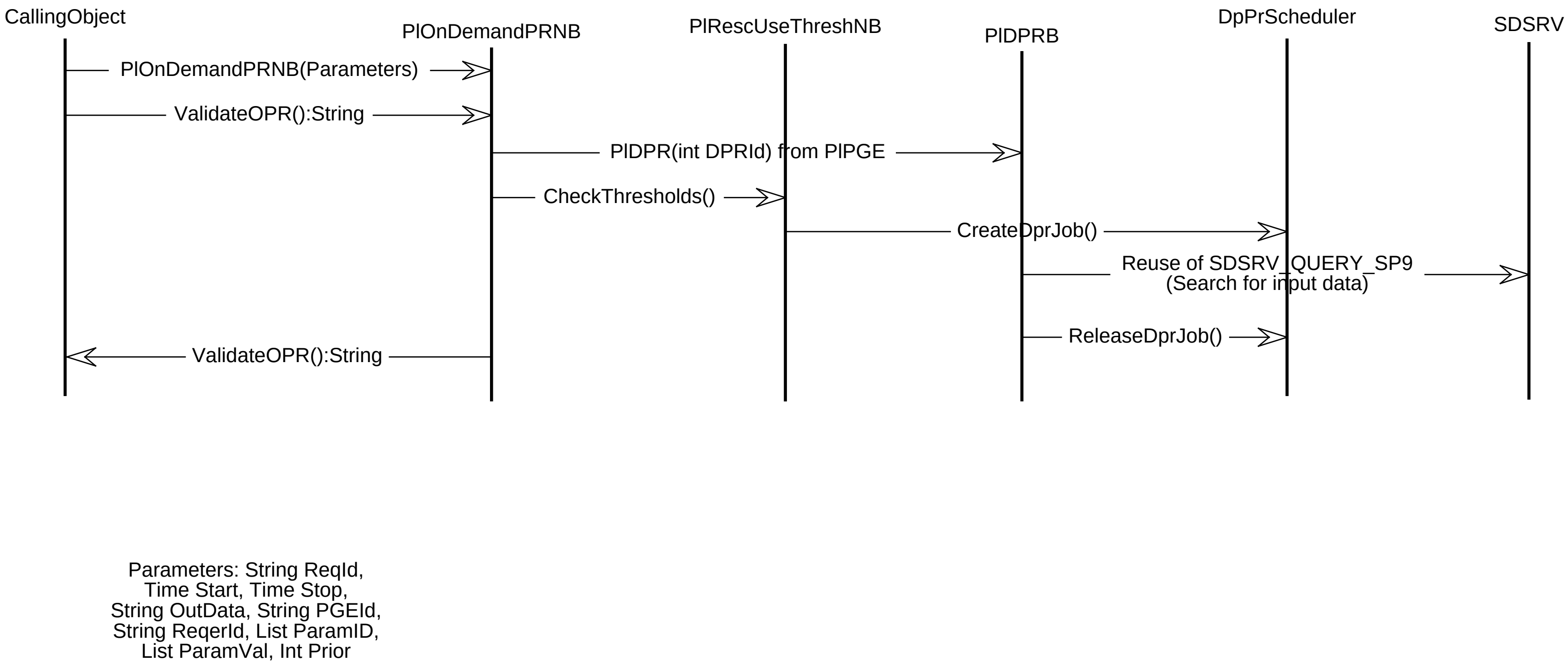

Figure 4-25. SUBMIT_OPR_SP49_B

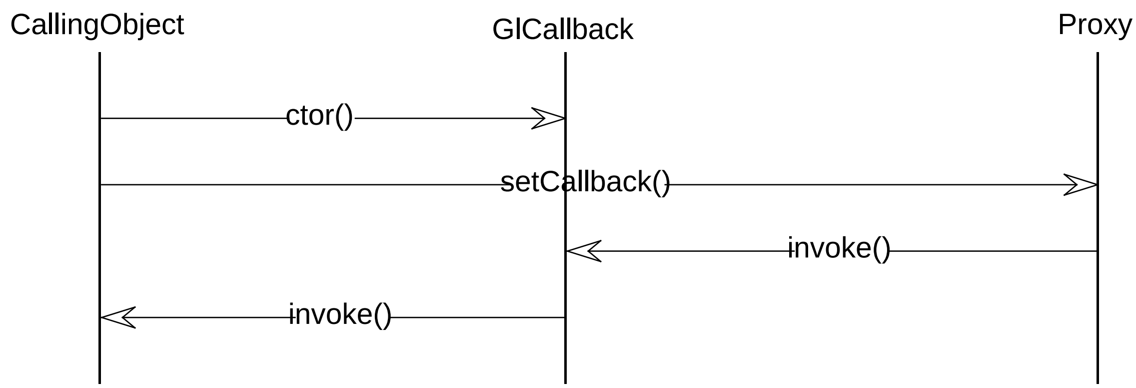

Figure 4-26. CALLBACK_SP50_B

4.2.28 DDICT_QUERY_SP51_B

This scenario primitive (see Figure 4-27) shows how the data dictionary is queried by a client. This scenario primitive involves two CSCIs: A requesting CSCI (Release B Client) and DDICT, which provides locations to which queries can be sent.

The calling object initiates a session with the DDICT by creating a DmDdRequestServer_C object. The DmDdRequestServer_C object is a synchronous session object which establishes a connection to a server side object DmDdRequestServer_S. The calling object calls the NewSearchRequest operation of DmDdRequestServer_C. The EcCsMsgHandler object of the Server Request Framework (SRF) creates a DmDdSearchRequest_C object upon notification that the correlated server side object has been created. The DmDdSearchRequest_C object is an asynchronous object that inherits from the SRF. The client must supply a callback function that will be called to inform the client of status from the server. When the client detects through the status parameter that the request is complete, the client will request a pointer to the result set using the GetResultSet operation of the DmDdSearchRequest_C object. The client will then interact with the DmDdResultSet operation to retrieve references to the results.

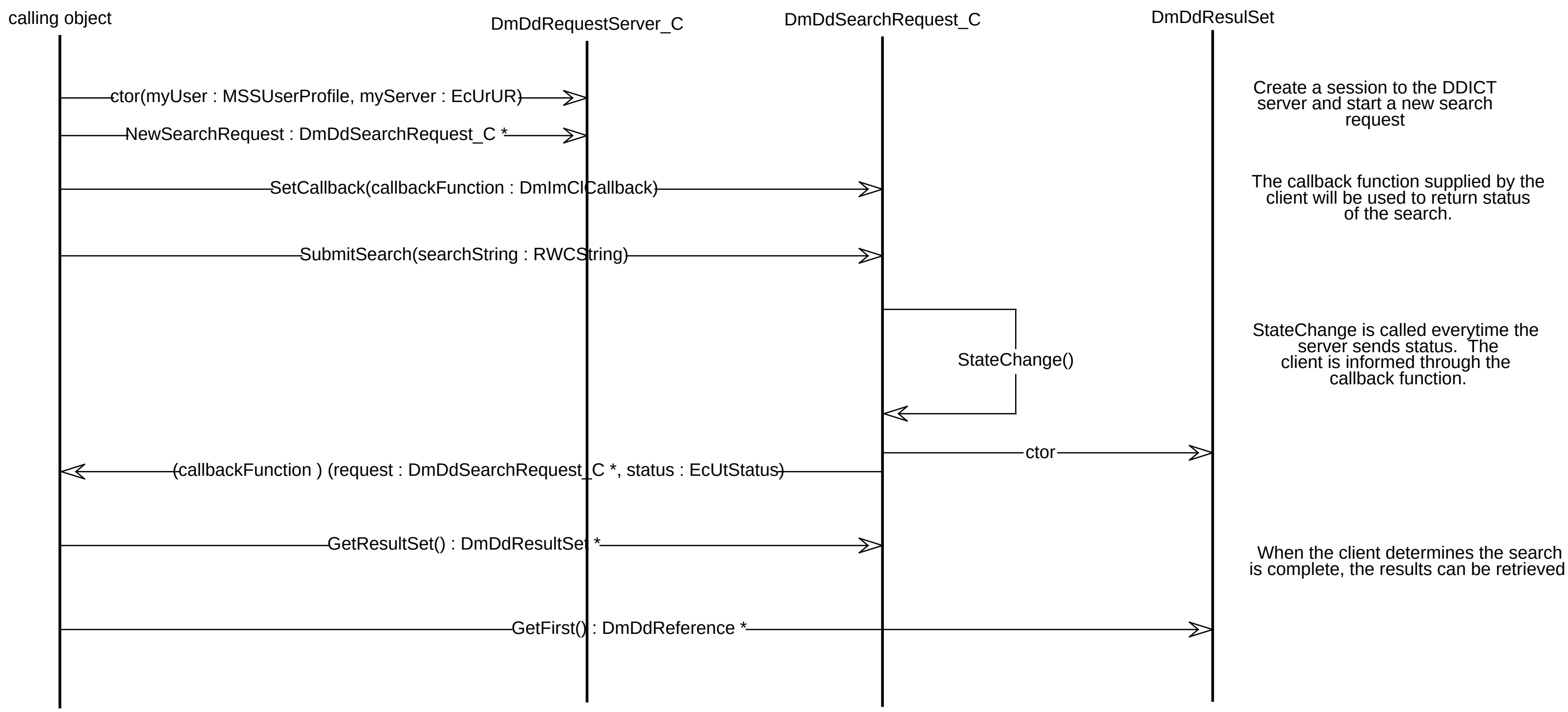

Figure 4-27. DDICT_QUERY_SP51_B

4.2.29 DIMGR_QUERY_SUBMISSION_SP52_B

This scenario primitive (see Figure 4-28) shows how a client submits a search request to the DIMGR. This scenario primitive involves two CSCIs: A requesting CSCI (Release B Client) and DIMGR.

The calling object (some object in the requesting CSCIs calling space) initiates a session with the DIMGR by creating a DmImClRequestServer object. The DmImClRequestServer object is a generic request factory object which establishes a connection to a server and instantiates a server-side request factory object. The server connection is established to the server with the UR specified in the constructor. The calling object initiates the creation of a new request by using the method NewRequest from the DmImClRequestServer object. The DmImClRequest object is then created using the callback principle of the Session Request Framework (SRF). The Calling Object populates the DmImClRequest with the search constraints. The calling object must specify a callback function to the request object so that the calling object can be notified of the completion of the request. Lastly, the calling object submits the request.

4.2.30 DIMGR_RESULT_RETRIEVAL_SP54_B

This scenario primitive (see Figure 4-29) shows how the Client retrieves data from the DIMGR, after the DIMGR has merged data results from the SDSRVs involved.

This scenario assumes that the following preconditions exist:

1. The server was constructed
2. A search request was initiated
3. A callback was set by the calling object
4. A search was submitted

When the search is complete, the following is what happens when the search is complete:

The calling object's callback is invoked upon completion of the data search. The calling object then specifies the range of results it can accommodate and retrieves the results specified from the DmImClRequest object. For example, the client can specify that it wants results 1 through 100 returned to reduce the amount of data being returned to the user. The return type is a GlParameterList that contains the attributes as requested in the query.

4.2.31 INSPECT_GRANULE_VALUE_PARAMETERS_SP55_1B

This scenario primitive (see Figure 4-30) shows one way SDSRV client applications can use the public SDSRV interface to get the value of parameters of a granule. This assumes that the client application has connected to the SDSRV and completed a search for granules. First the client application goes to the first granule in its list. Then it needs to build a GlParameterList of all the parameters that it wants values for. In this example we are assuming that the client is interested in the Sensor of the granules. Once the list of parameters of interest is built, the list is given to the Inspect service of the granule of interest. The list of parameters will be populated for the client application. Then the client application can simply get the values of the parameters. This sequence can easily be repeated for all granules in the DsClESDTReferenceCollector, as the DsClESDTReferenceCollector uses the RogueWave Collectable interface.

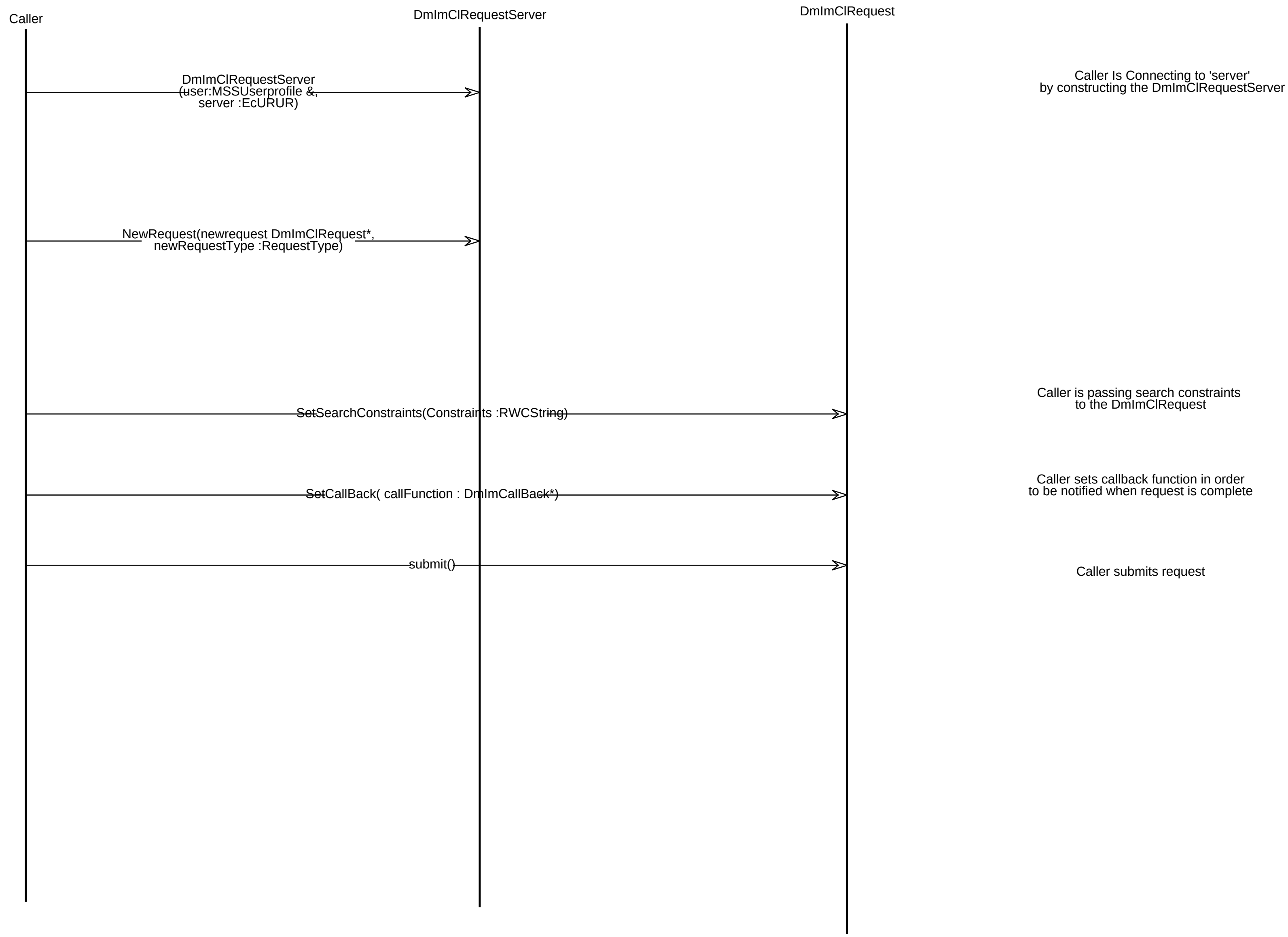

Figure 4-28. DIMGR_QUERY_SUBMISSION_SP52_B_B

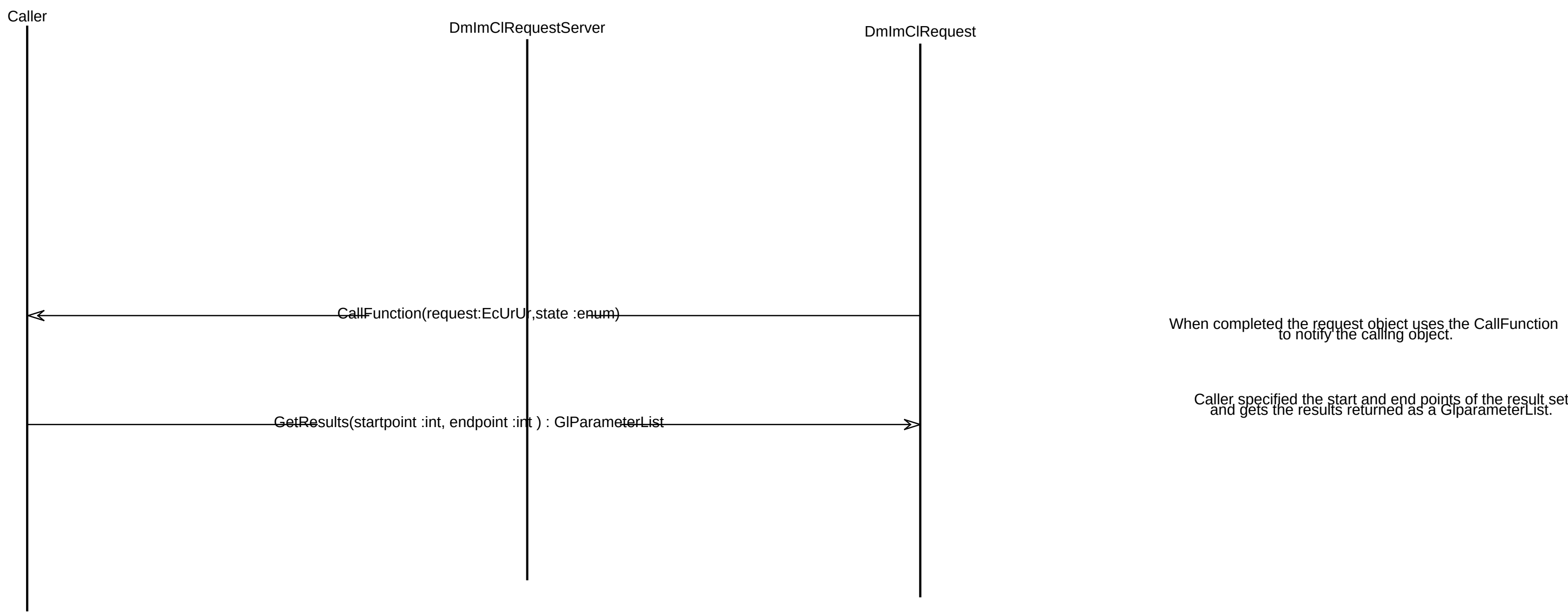

Figure 4-29. DIMGR_RESULT_RETRIEVAL_SP54_SP54_B

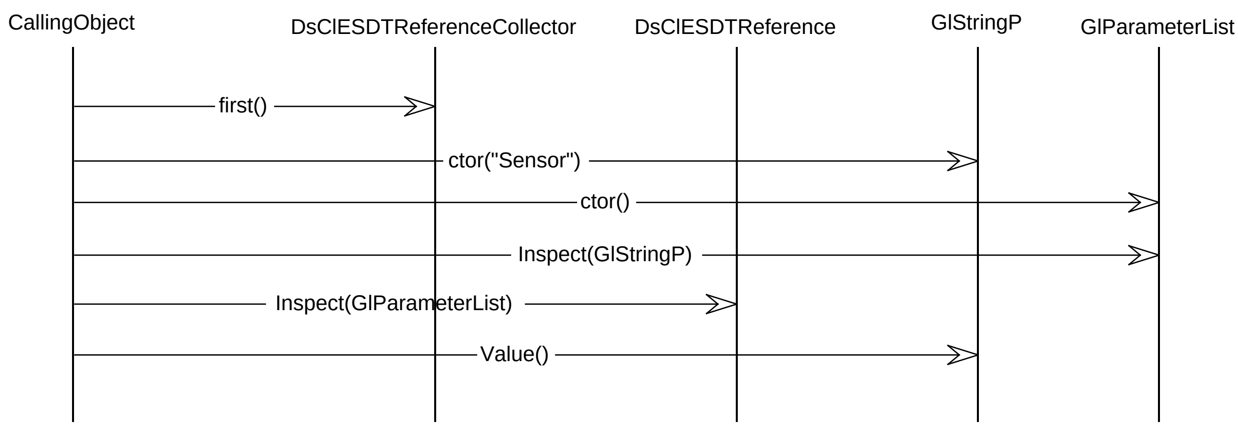

Figure 4.-30. INSPECT_GRANULE_VALUE_PARAMETERS_SP55_1B Diagram

4.2.32 INSPECT_GRANULE_UR_SP55_2B

This scenario primitive shows how a SDSRV client application can get the UR of a granule from a DsClESDTReference object. This assumes that the client application has connected to the SDSRV and has the DsClESDTReference object in its collection. The SDSRV client first gets to the granule of interest, in this case assumed to be the first one on the list. Then the GetUr method will return a reference to the GranuleUr, which is derived from the base UR Infrastructure.

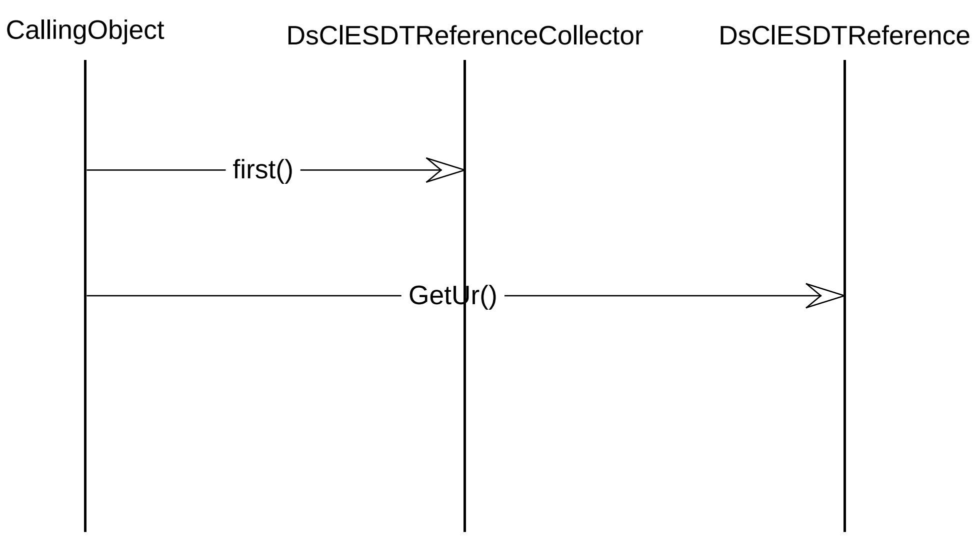

Figure 4-31. INSPECT_GRANULE_UR_SP55_2B Diagram

4.2.33 ACQUIRE_DATA_WITH_UR_SP56_B

This scenario (see Figure 4-32) shows how a request is made to acquire data from the SDSRV. The scenario involves two CSCIs: the requesting CSCI (the owner of the CallingObject); and SDSRV, which provides the data.

The main interface with SDSRV is through an instance of the SDSRV distributed class, DsClESDTReference, which has been provided the identity of a GlCallback instance to be called upon completion of the request. An instance of DsClRequest, which is imported but not distributed, is created and one or more instances of DsClCommand are inserted into it. Into each DsClCommand object is inserted a list of parameters which define the data to be acquired, and an attribute of DsClCommand is set to indicate that it is an “Acquire” command. The calling object then asks each request to submit itself. Each DsClRequest instance then inserts itself into the DsClESDTReference instance, which then submits the requests to SDSRV. The requesting CSCI continues after making this synchronous call. Later, SDSRV notifies the requesting CSCI of the completion status by calling the invoke operation on the specified GlCallback instance. The call to GlCallback is a local call from a distributed SDSRV object.

GlCallback is not implemented as an object but implemented as a typedef called DsTClRequestCallback that is a function pointer to an entry point in the client application.

4.2.34 INGEST_ADSRV_INTERFACE_SP57_B

The following scenario primitive (see Figure 4-33) provides an overview for how the Ingest subsystem populates the Ingest data type data base with service universal references (URs) for the distributed SDSRV and STMGT services assigned to the archive all valid Ingest data type (i.e. CERES02).

Once populated the data type template information will be accessed by Ingest Data Server Insertion software (see General Data Ingest scenario section 4.1.5), and used to send resource allocation requests to STMGT and data archive requests to SDSRV.

The Ingest operator initiates this processing by selecting the appropriate GUI option from the Monitor and Control GUI screen. This instantiates a GUI session which executes an InDBAccess operation.

The InDBAccess function extracts each ingest data type defined in InDataType template to IoAdProduct Search command to get corresponding SDSRV and STMGT URs.

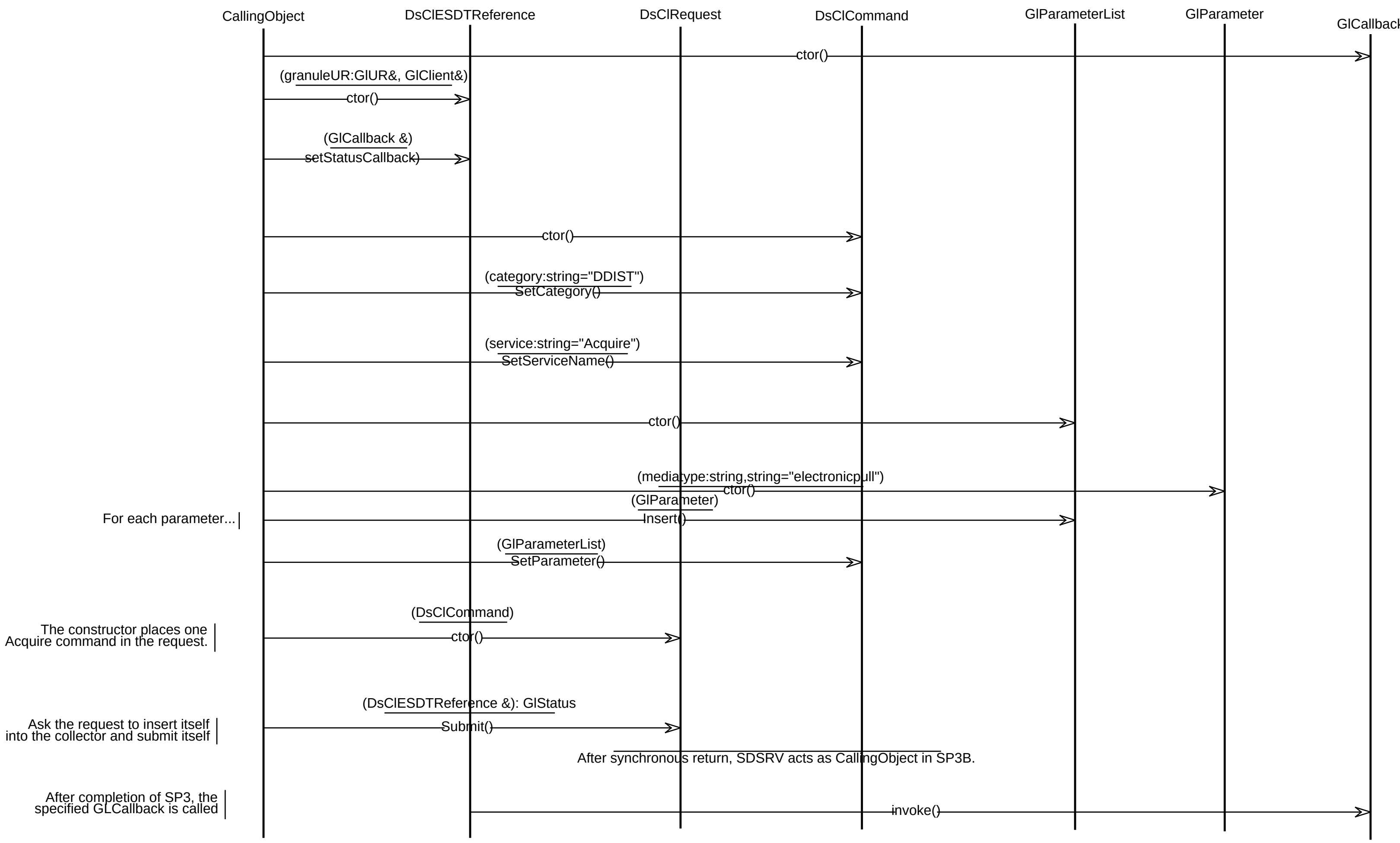

Figure 4-32. ACQUIRE_DATA_WITH_UR_SP56_B Diagram

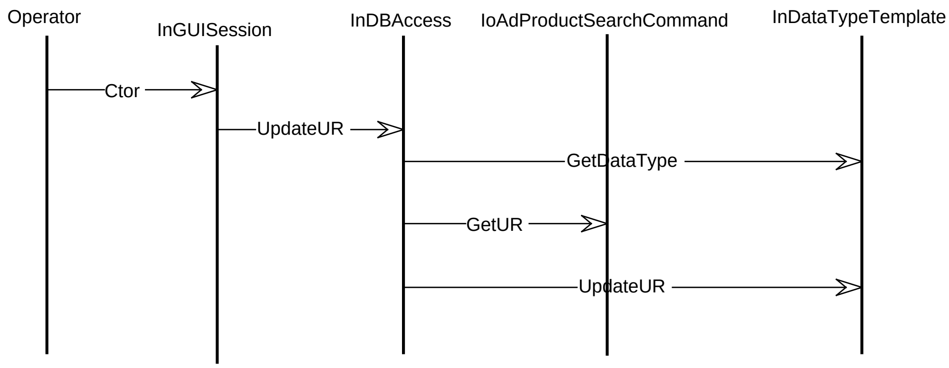

Figure 4-33. INGEST_ADSRV_INTERFACE_SP57_B Diagram

4.2.35 RETRIEVE_PRODUCT_PRC_SP58_B

This scenario (see Figure 4-34) shows how a price estimate for a request is made from SDSRV to the MSS Billing and Accounting Application Service (BAAS). The scenario involves three CSCIs: the requesting CSCI (the owner of the CallingObject); SDSRV, which provides the data request identifying information; and CSMS, which provides the standard pricing information used in generating an estimate for the SDSRV data acquire.

The calling object within SDSRV creates a request that represents a billable data product. If the original request contains an acquire of multiple billable data products, the request is broken down into subrequests that can be estimated based on an identifiable, priceable item and tracked by requestId. In addition to an identifiable product, any services required to process the product, the media type, number of media required to fufill the request and shipping method used, will be required in a global parameter list, GlParameterList, in order to price the request accurately. The price of the product request is then retrieved from the standard price table, EcPriceTableB, which returns to the requesting object this price estimate by invoking the operation ProvidePrices().

4.2.36 VERIFY_AVAILABLE_FUNDS_SP59_B

This scenario (see Figure 4-35) shows how a science user's user profile balance is checked in preparation for authorizing a data product request from SDSRV. After a price estimate for a request is made from SDSRV to the MSS Billing and Accounting Application Service (BAAS), the user's available balance will be checked and compared to the amount of the request estimate.

The scenario involves two CSCIs: the SDSRV, which provides the user identifying information and the BAAS which provides the available user profile balance.

The calling object from SDSRV will request the available balance from a client instance of MsAcUsrProfileMgr, given the requester's userId by retrieving the science user's profile. A manager class for the user profile information is in place to ensure data integrity of sensitive information. The client instance of MsAcUsrProfileMgr will request that the server profile manager persistence instance get the science user's current account balance from the ECS Management database. The server instance of MsAcUsrProfileMgr will send the user's available balance to the client side which will populate (set) the account balance field in the current instance of the MsAcUsrProfile class. The calling object from SDSRV will use the method GetAccountBalance() with the current userId to retrieve the balance so that the available funds for this user can be verified.

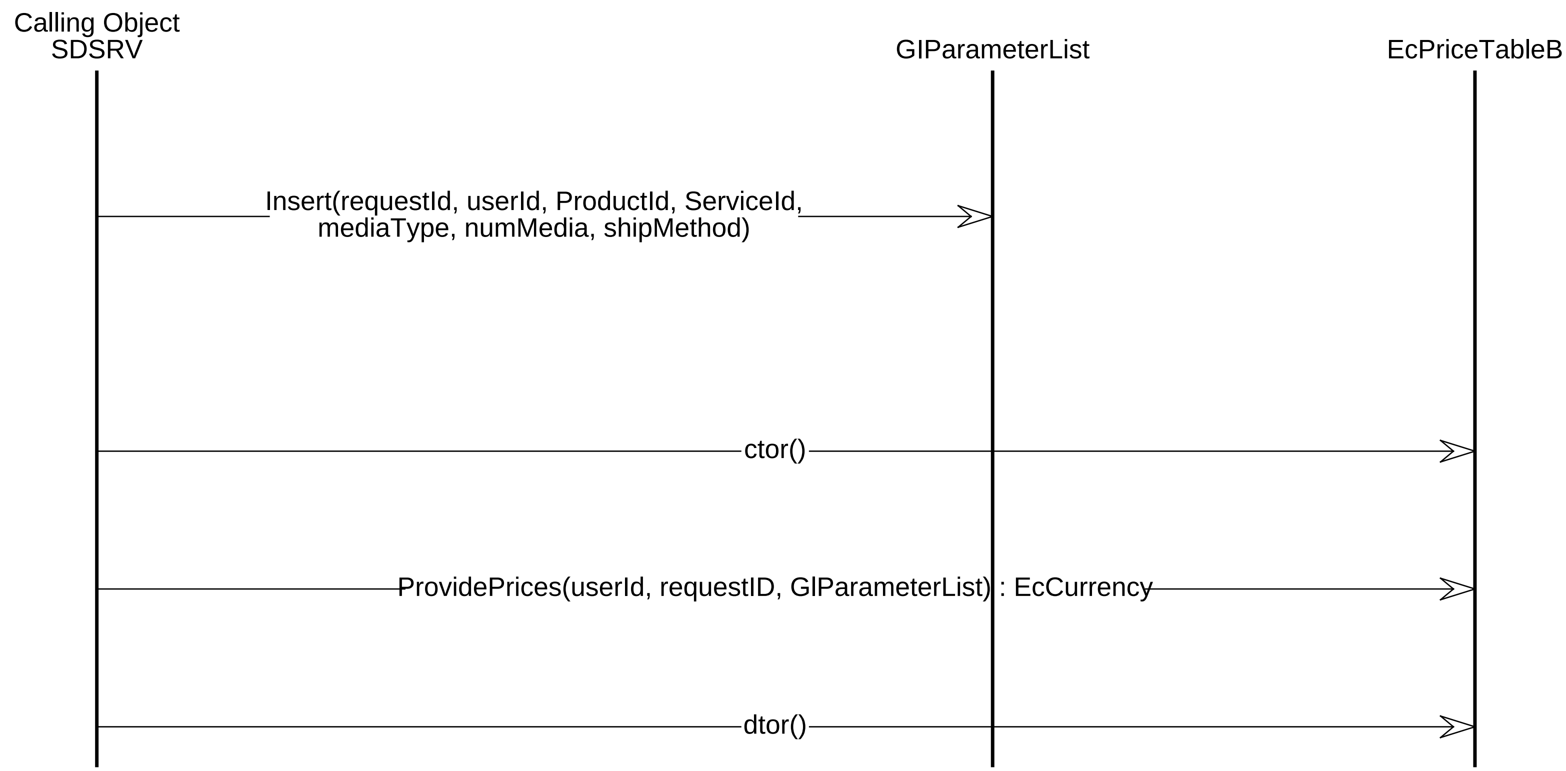

Figure 4-34. RETRIEVE_PRODUCT_PRC_SP58_B Diagram

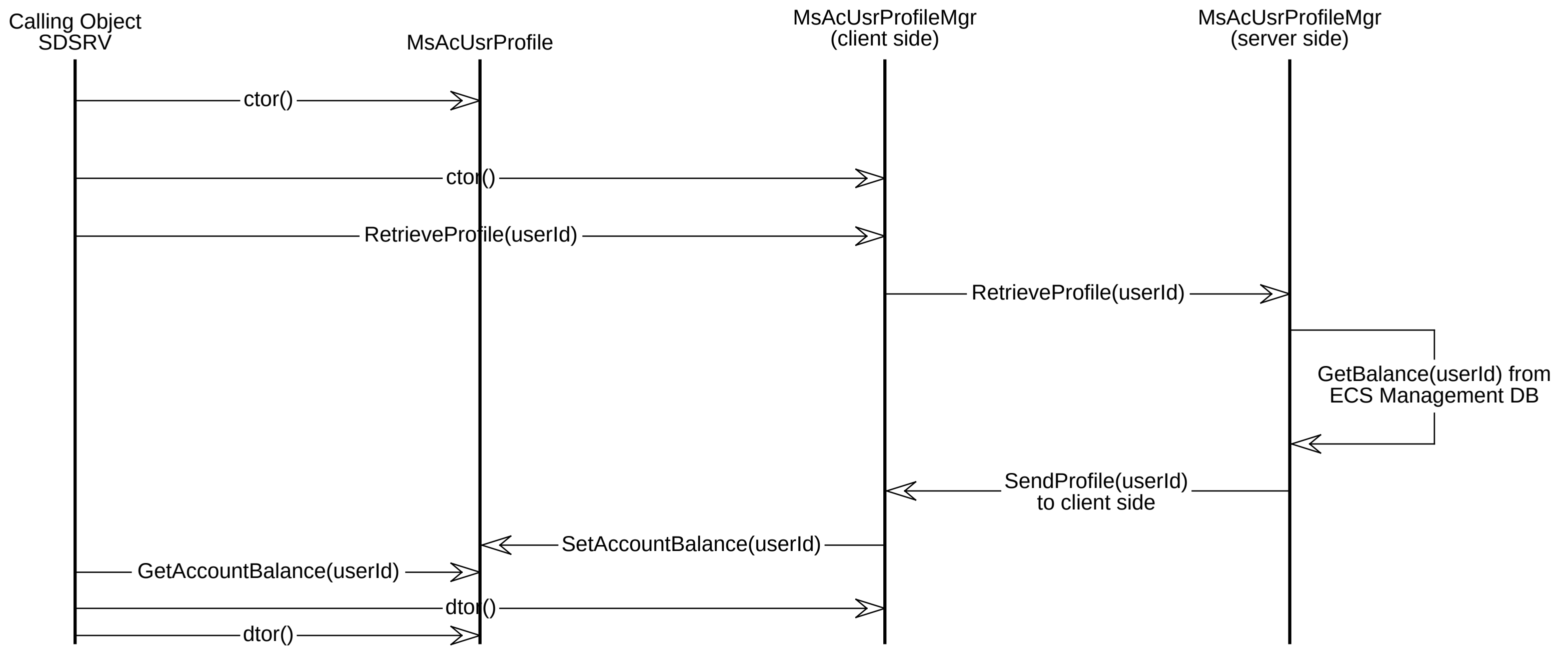

Figure 4-35. VERIFY_AVAILABLE_FUNDS_SP59_B Diagram

4.2.37 UPDATE_USER_PROFILE_SP60_B

This scenario (see Figure 4-26) shows how a the science user's user profile balance is debited for the amount authorized in a pending data product request that has already been priced by the MSS Billing and Accounting Application Service (BAAS) and is ready to ship.

The scenario involves two CSCIs: SDSRV, which provides the user identifying information and CSMS which provides the available user profile balance to determine if sufficient funds exist to honor the completion and delivery of the request.

SDSRV submits a request to deduct the amount of a confirmed data product request from the balance of the science user. This amount is submitted to the MSS accountability tracking manager object, MsAcTrackingMgr. A client object instance of the MsAcProfileMgr is created which will retrieve the user's profile information from a persistence instance of the user profile manager on the server side. MsAcUsrProfileMgr (server) will get the balance from the ECS Management database and return this information to the client side. An instance of the current user's profile will be created, and then populated (set) by the MsAcUsrProfileMgr. Client/server object instances for the MsAcProfileMgr are used to promote data integrity of user's sensitive information, such as their account balance. The tracking manager, MsAcTrackingMgr then invokes the GetAccountBalance() method with the current userId to get the recently populated MsAcUsrProfile balance. A new balance is then calculated taking into account the price of the request which was computed in Scenario Primitive SP 58B, Retrieve Product Price. The balance in the MsAcUsrProfile is adjusted by invoking the method SetAccountBalance() by debiting the amount of the product estimate, signaling to the SDRSV calling object that the request can be honored.

4.2.38 SUBMIT_SERVICE_AD_SP61_B

This scenario primitive (see Figure 4-37) demonstrates how to advertise a service (i.e. how to submit a Service advertisement to the ADSRV). This scenario primitive involves two CSCIs: a requesting CSCI and ADSRV.

The calling object (some object in the requesting CSCIs calling space) creates an IoAdProvider object, which acquires the provider for the service from the database. The calling object then creates an IoAdContact object. The calling object calls the IoAdContact object to submit the necessary information regarding the contact person or organization, such as last name, first name, address, phone number, email, etc., who is responsible for this Service advertisement. The provided information is then stored in the database. The calling object then creates an IoAdService object. The calling object submits information, such as ServiceClass, ServiceName, ServiceTypeId, to the IoAdService object. The calling object then calls the IoAdService object again to store the provided information into the database. The Store method returns a status describing whether the operation completed without error.

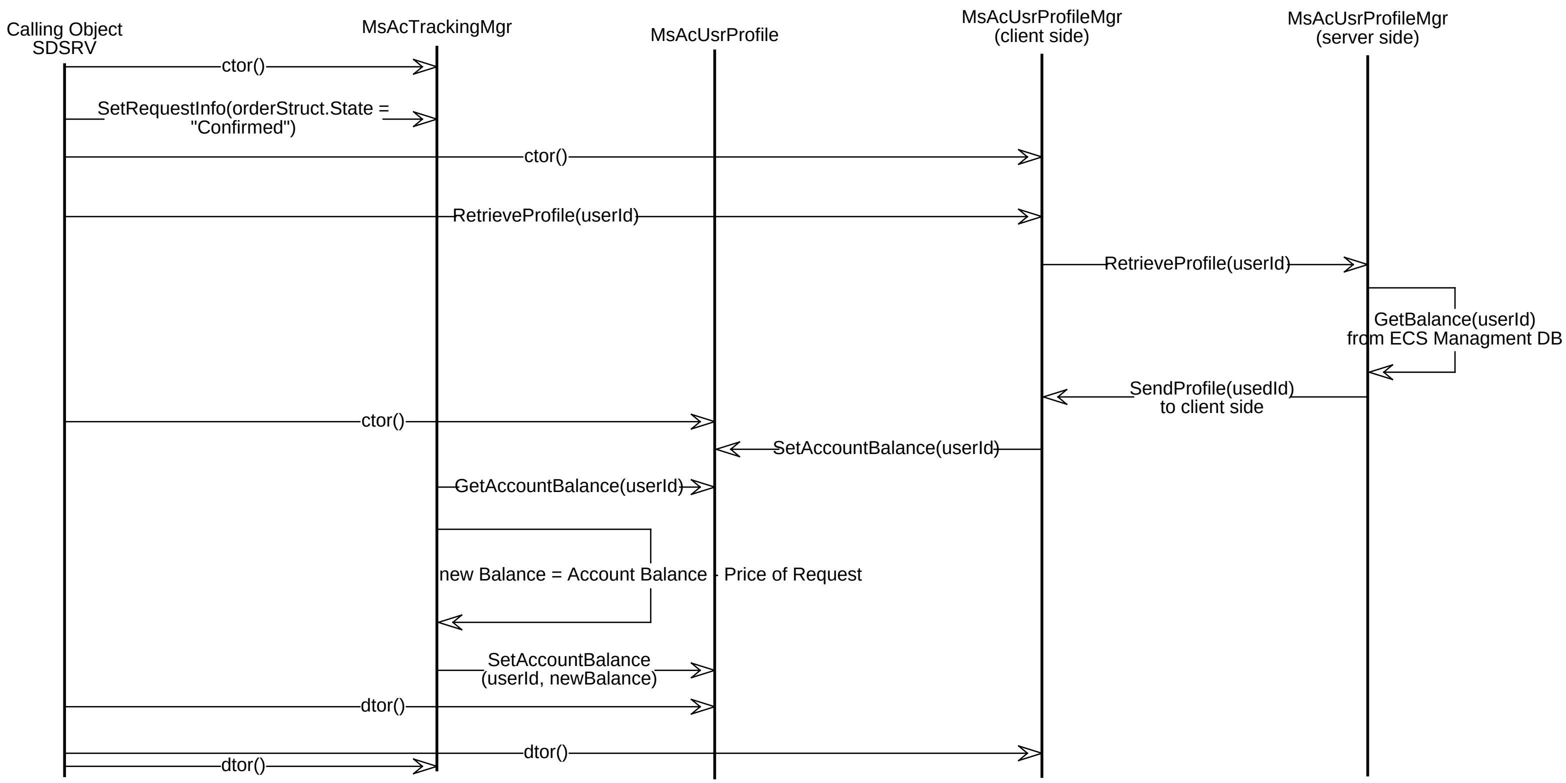

Figure 4-36. UPDATE_USER_PROFILE_SP60_B Diagram

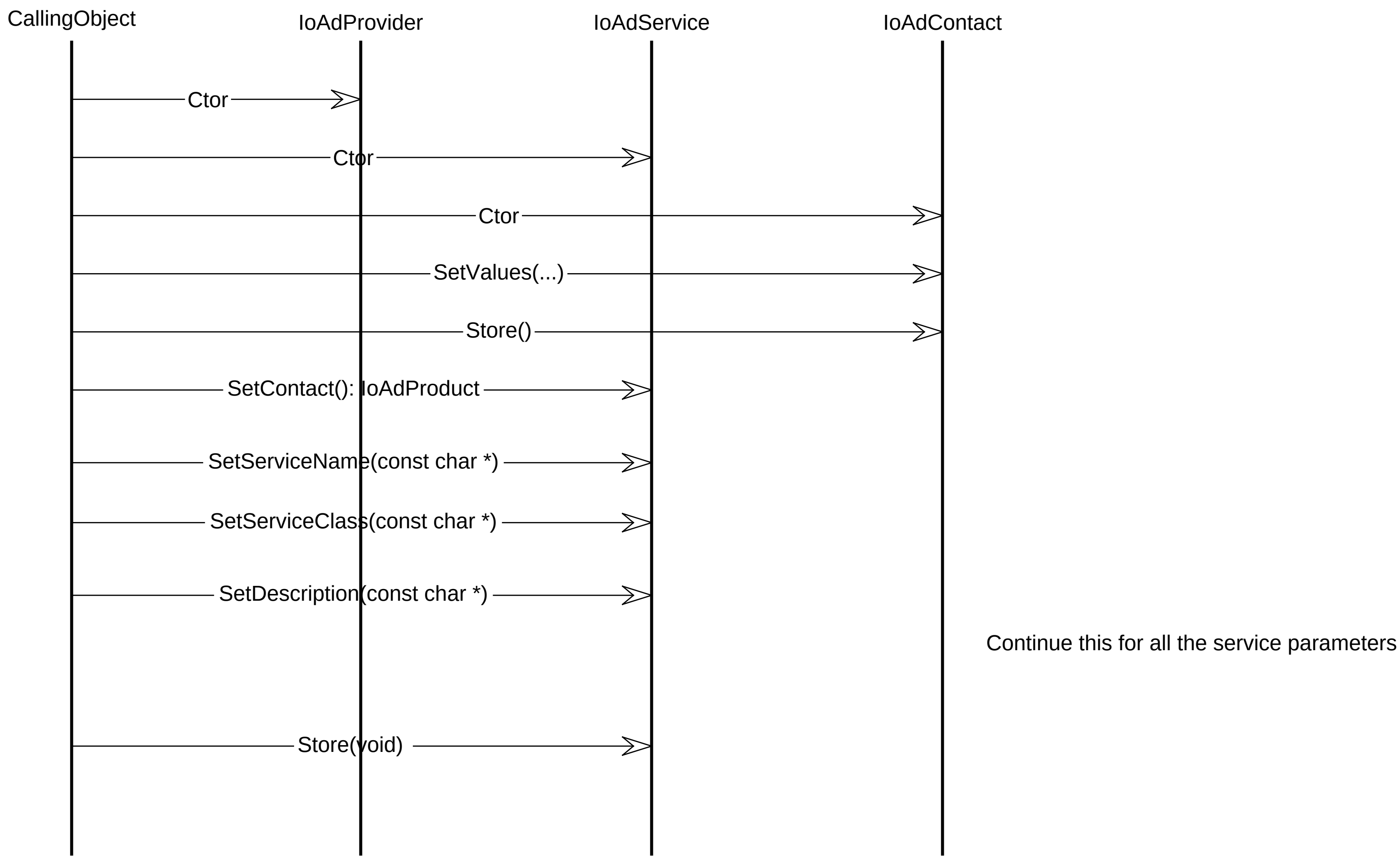

Figure 4-37. SUBMIT_SERVICE_AD_SP61_B Diagram